Structural Kernel Search via Bayesian Optimization and Symbolical Optimal Transport

Matthias Bitzer **Mona Meister** **Christoph Zimmer**
Bosch Center for Artificial Intelligence, Renningen, Germany
{matthias.bitzer3,mona.meister,christoph.zimmer}@de.bosch.com

Abstract

Despite recent advances in automated machine learning, model selection is still a complex and computationally intensive process. For Gaussian processes (GPs), selecting the kernel is a crucial task, often done manually by the expert. Additionally, evaluating the model selection criteria for Gaussian processes typically scales cubically in the sample size, rendering kernel search particularly computationally expensive. We propose a novel, efficient search method through a general, structured kernel space. Previous methods solved this task via Bayesian optimization and relied on measuring the distance between GP's directly in function space to construct a kernel-kernel. We present an alternative approach by defining a kernel-kernel over the symbolic representation of the statistical hypothesis that is associated with a kernel. We empirically show that this leads to a computationally more efficient way of searching through a discrete kernel space.

1 Introduction

In many real-work applications of machine learning, tuning the hyperparameters or selecting the machine learning method itself is a crucial part of the workflow. It is often done by experts and data-scientist. However, the number of possible methods and models is constantly growing, and it is becoming increasingly important to automatically select the right model for the task at hand. Bayesian optimization (BO) is a prominent method that can be used for model selection and hyperparameter tuning. It can handle black-box oracles with expensive function evaluations, which are two characteristics often encountered when doing model selection [17]. Important applications in this context are choosing the hyperparameters in the training process of deep neural networks (DNN) [17], or dealing with discrete and structured problems like choosing the architecture of DNN's [5, 11].

Gaussian processes (GP) are another important model-class. They are often utilized as surrogate models in BO [17], for time-series and statistical modeling [7, 3] or in active learning loops [23, 15]. The properties of GP's are mainly governed by its kernel that specifies the assumptions made on the underlying function. Choosing the right kernel is therefore a crucial part of applying GP's and is often done by the expert. Recent work [9] treated the kernel selection as a black-box optimization problem and used Bayesian optimization to solve it. This allowed searching over a highly structured, discrete space of kernels. However, their proposed kernel-kernel measures the distance between two GP's directly in function space, which is a computationally expensive task itself. This makes the method difficult to apply for the frequent scenarios where the evaluation of the model selection criteria requires only a medium amount of time.

We propose measuring the distance between two kernels via their symbolical representation of their associated statistical hypothesis. We utilize the highly general kernel-grammar, presented in [3], as underlying kernel space, where each kernel is build from base kernels and operators, like e.g.

$$\mathrm{LIN} + ((\mathrm{SE} \times \mathrm{PER}) + \mathrm{SE})$$

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

forming effectively a description of the statistical hypothesis that is modeled by the GP. Our main idea is to build a distance over these symbolical descriptions, rather than measuring the distance between two GP's directly in function space. We employ optimal transport principles, known from neural-architecture search (see [11],[5]), to build a pseudo-distance between two hypotheses descriptions and use it to construct a kernel-kernel, which is subsequently utilized in the BO loop. We will show that the induced kernel search method is more efficient, in terms of number of function evaluations and computational time, compared to alternative kernel search methods over discrete search spaces.

The main challenge we encountered is the quantification of dissimilarity between two symbolical representations of kernels. We use the tree representation of each symbolical description and apply optimal transport distances over tree features. Subsequently, we empirically show that the deduced GP over GP's provides a well-behaved meta-model and show its advantages for kernel search. In summary, our contributions are:

1. We construct a pseudo-distance over GP's that acts over the symbolical representations of the underlying statistical hypothesis.
2. We use the pseudo-distance to construct a novel "kernel-kernel" and build it in a BO loop to do model selection for GP's.
3. We empirically show that our meta-GP model is well-behaved and that we outperform previous methods in kernel search over discrete kernel spaces.

Related Work: There is considerable existing work on constructing flexible kernels and learning their hyperparameters [1, 20, 21, 18, 22]. In these kinds of works, the structural form of the kernel is predefined and the free parameters of the kernel are optimized, often via marginal likelihood maximization. While being able to efficiently finding the hyperparameter due to the differentiability of the marginal likelihood, one still needs to predefine the structural form of the kernel in the first place. This is a hard task as one need to decide if long-range correlation or nonstationarity should be considered or if dimensions are ignored or not. Our method is build to automatically select the structure of the kernel. Some of the mentioned methods [20, 21] are able to approximate any stationary kernel via Bochner's Theorem and therefore consider a broad kernel space themselves. However, even elementary statistical hypotheses require nonstationary kernels, such as linear trends modeled by the linear kernel. Our search space is not restricted to stationary kernels.

We consider a highly general, discrete kernel space that is induced by the kernel grammar [3]. Recent work [9] used BO to search through this space via a kernel that measures similarity in function space. We also utilize BO but employ a fundamentally different principle of measuring the distance, which is computationally more efficient. We dedicate Section 3.2 to a more precise comparison to the method of [9]. Additionally, the original kernel grammar paper [3] suggested greedy search for searching through the kernel grammar. Furthermore, [4] employed a genetic algorithm based on cross-over mutations. We empirically compare against all approaches.

In the area of BO over structured spaces, our method is most similar to the neural-architecture search (NAS) procedures presented in [11, 5] who use optimal transport distances over features extracted from the graph-representation of neural networks. Compared to these methods we use OT principles to do model selection for GP's which is a fundamentally different task.

2 Background and Set-up

Our main task is efficient model selection for Gaussian processes. In order to provide background information, we give a small introduction to GP's and model selection for GP's. Subsequently, we present a review of the kernel-grammar [3] and show how we use it for our approach. We show how a kernel can be represented via a symbolical description. In Section 3, we will present how we use the symbolical description to construct a kernel over kernels and how we use it in the BO loop.

2.1 Gaussian Processes

A Gaussian process is a distribution over functions $f : \mathcal{X} \to \mathbb{R}$ over a given input space $\mathcal{X}$ which is fully characterized via the covariance/kernel function $k(x, x') = \mathbf{Cov}(f(x), f(x'))$ and the mean function $\mu(x) := \mathbb{E}[f(x)]$. We therefore can write as shorthand notation $f \sim \mathcal{GP}(\mu(\cdot), k(\cdot, \cdot))$. The

kernel can be interpreted as a similarity measure between two elements of the input space, with the GP assigning higher correlations to function values whose inputs are more similar according to the kernel. Furthermore, the kernel governs the main assumptions on the modeled function such as smoothness, periodicity or long-range correlations and therefore provides the inductive-bias of the GP. An important property of GP's is that they are not restricted to euclidean input spaces $\mathcal{X} \subset \mathbb{R}^d$, but can also be defined on highly structured spaces like trees and graphs, a property we will later use to define a GP over GP's.

While our method might also be used for model selection in classification, we consider from now on Gaussian processes regression. For regression, a dataset $\mathcal{D} = (\mathbf{X}, \mathbf{y})$ with $\mathbf{X} = \{x_1, \dots, x_N\} \subset \mathcal{X}$ and $\mathbf{y} = (y_1, \dots, y_N)^\intercal \in \mathbb{R}^N$ is given, where we suppose that $f \sim \mathcal{GP}(\mu(\cdot), k(\cdot, \cdot))$ and $y_i = f(x_i) + \epsilon_i$ with $\epsilon_i \overset{i.i.d}{\sim} \mathcal{N}(0, \sigma^2)$. Given the observed data $\mathcal{D}$ the posterior distribution $f|\mathcal{D}$ is again a GP with mean and covariance functions

$$\mu_{\mathcal{D}}(x) = \mu(x) + \mathbf{k}(x)^\intercal(\mathbf{K} + \sigma^2 I)^{-1}(\mathbf{y} - \mu(\mathbf{X})),$$
$$k_{\mathcal{D}}(x, y) = k(x, y) - \mathbf{k}(x)^\intercal(\mathbf{K} + \sigma^2 I)^{-1}\mathbf{k}(y)$$

with $\mathbf{K} = [k(x_m, x_l)]_{m,l=1}^N$ and $\mathbf{k}(x) = [k(x, x_1), \dots, k(x, x_N)]^\intercal$ (see [13]). Probabilistic predictions can be done via the resulting predictive distribution $p(f^*|x^*, \mathcal{D}) = \mathcal{N}(\mu_{\mathcal{D}}(x^*), k_{\mathcal{D}}(x^*, x^*))$.

2.2 Model selection for GP's

Typically, the kernel k_θ comes with a set of parameters θ that can be learned via maximization of the marginal likelihood $p(\mathbf{y}|\mathbf{X}, \theta, \sigma^2) = \mathcal{N}(\mathbf{y}; \mu(\mathbf{X}), k_\theta(\mathbf{X}, \mathbf{X}) + \sigma^2\mathbf{I})$ or via maximum a posteriori (MAP) estimation, in case the parameters θ are equipped with a prior $p(\theta)$. This procedure is sometimes also called model selection, as one selects the hyperparameters of the kernel given the data (see [13]). However, we consider selecting the *structural form* of the kernel itself. The structural form of the kernel determines the statistical hypothesis that is assumed to be true for the data-generating process. Intuitively, the kernel is similar to the architecture in deep neural networks, which induces an inductive bias.

Our goal is to do model selection over a discrete, possibly infinite space of kernels $\mathbb{K} := \{k_1, k_2, \dots\}$. As each kernel comes with its own parameters, we are actually dealing with a space of kernel families. Thus, when mentioning a kernel k we associate it with its whole family over parameters $\{k_\theta | \theta \in \Theta\}$. Once a kernel is selected, predictions are done with learned kernel parameters (that usually are a by-product of calculating the model selection criteria). The parameters θ are potentially equipped with a prior $p(\theta)$ depending on the selection criteria. As mean function, we always utilize the zero mean function $\mu(x) := 0$ in case $\mathbf{y}$ is centered and a constant mean function otherwise. This is a common choice in GP regression. Given some model selection criteria $g : \mathbb{K} \to \mathbb{R}$ our task is solving $k^* = \arg\max_{k \in \mathbb{K}} g(k|\mathcal{D})$. While our method is not restricted to a specific model selection criteria, we focus on the model evidence $p(\mathbf{y}|\mathbf{X}, k)$, which is a well-known selection criteria for probabilistic models (see [9, 14, 8]). Given a prior on the kernel parameters $p(\theta)$ and the likelihood variance $p(\sigma^2)$, the log-model evidence of the marginalized GP is given as

$$g(k|\mathcal{D}) = \log p(\mathbf{y}|\mathbf{X}, k) = \log \int p(\mathbf{y}|\mathbf{X}, \theta, \sigma^2, k) p(\sigma^2) p(\theta|k) d\theta d\sigma^2.$$

This quantity can be approximated via Laplace approximation of $p(\theta, \sigma^2|\mathcal{D})$ (see Appendix A for details). Computing this approximation includes performing a MAP estimation of the GP parameters. Thus, once the log evidence has been computed, learned kernel hyperparameters θ_{MAP} are automatically provided. Performing the MAP estimation scales cubically in the data set size N, which renders model selection for GP's computationally intense.

2.3 Kernel Grammar

The kernel grammar introduced by [3] specifies a highly general search space over kernels. The grammar is based on the observation that kernels are closed under addition and multiplication, meaning, for kernels $k_1(x, x')$ and $k_2(x, x')$ also $k_1(x, x') + k_2(x, x')$ and $k_1(x, x') \times k_2(x, x')$ are kernels. Given some base kernels, such as the squared exponential kernel SE, the linear kernel LIN, the periodic kernel PER or the rational quadratic kernel RQ, different statistical hypotheses can be

generated via addition and multiplication. For example,

$$\text{LIN} + \text{PER} \times \text{SE}$$

describes a linear trend with a locally periodic component, such that it might be a useful hypothesis for time-series applications. For multidimensional data the base kernels are applied to single dimensions, denoted e.g. with SE_i for the squared exponential kernel defined on dimension i. The grammar contains many popular hypotheses such as the ARD-RBF kernel that can be expressed via multiplication over the dimensions $\prod_{i=1}^{d} \text{SE}_i$ as well as additive models $\sum_{i=1}^{d} \text{SE}_i$ or polynomials of order m with $\prod_{j=1}^{m} \text{LIN}$.

The specification of the complete search space is given as a grammar, where a base kernel is denoted as $\mathcal{B}$ and a subexpression is denoted with $\mathcal{S}$. For example, in the expression $\text{LIN} + (\text{PER} \times \text{SE})$ the expression $\text{PER} \times \text{SE}$ is a subexpression. Starting with all base kernels, the search space is defined as all kernels that can be reached via the following operations:

1. Add a base kernel to a subexpression: $\mathcal{S} \to \mathcal{S} + \mathcal{B}$
2. Multiply a subexpression with a base kernel: $\mathcal{S} \to \mathcal{S} \times \mathcal{B}$
3. Exchange a base kernel with another base kernel: $\mathcal{B} \to \mathcal{B}'$

Starting from the set of base kernels, these operations can lead to all algebraic expressions, showing the expressiveness of the grammar [3]. The authors of [3] suggest using greedy search to search through the grammar, where the kernel with the highest value for the selection criteria is selected and expanded with all possible operations. After the selection criteria is calculated on the neighbors, the search progresses to the next stage by expanding the neighbors of the best kernel found.

We consider a generalized notion of the kernel grammar, where we consider a set of base kernels $\{\mathcal{B}_1, \ldots, \mathcal{B}_r\}, r \in \mathbb{N}$ and a set of operators $\{T_1, \ldots, T_l\}, l \in \mathbb{N}$ where $T_j : \mathcal{K} \times \mathcal{K} \to \mathcal{K}, j = 1, \ldots, l$ are closed operators on the space of all kernel functions $\mathcal{K}$. Examples are addition and multiplication, but also the change-point operator which was considered in [7]. Thus, in general the grammar operations are

1. Apply operator T_j onto a subexpression and a base kernel[1]: $\mathcal{S} \to T_j(\mathcal{S}, \mathcal{B})$
2. Exchange a base kernel with another base kernel: $\mathcal{B} \to \mathcal{B}'$

The considered kernel space can be defined precisely in the following way:

Definition 1 *For $r, l \in \mathbb{N}$, let $\{k_1, \ldots, k_r\}$ be a set of (base-) kernels (symbollically represented as $\{\mathcal{B}_1, \ldots, \mathcal{B}_r\}$) and $\{T_1, \ldots, T_l\}$ a set of operators with $T_j : \mathcal{K} \times \mathcal{K} \to \mathcal{K}, j = 1, \ldots, l$. Let*

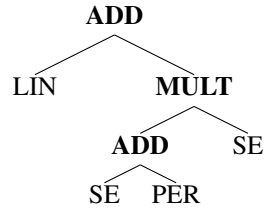
$$L_0 := \{k_1, \ldots, k_r\}$$
$$L_i := \{T_j(k_1, k_2) \mid k_1, k_2 \in L_{i-1}, j = 1, \ldots, l\} \cup L_{i-1}$$

for $i = 1, \ldots, M$. We call $\tilde{\mathbb{K}} := L_M$ the kernel-grammar generated kernel space with depth M.

2.4 Representation of Kernels

The base kernels and their combination via the operators describe the structural assumptions of the final kernel and imply the assumptions that are made in function space. Our main hypothesis is that the symbolical representation of the kernel, given by the subexpressions $\mathcal{S}$ and base kernels, already contains sufficient information for model selection. Therefore, we will define a kernel-kernel over the symbolical representations and utilize it for Bayesian optimization. Concretely, each kernel $k \in \tilde{\mathbb{K}}$ in our described space can be written as a tree $\mathcal{T}$, for example:

$$\text{LIN} + ((\text{SE} + \text{PER}) \times \text{SE}) \quad \longleftrightarrow$$



[1] In case the operator T_j is not symmetric we also add the operation $\mathcal{S} \to T_j(\mathcal{B}, \mathcal{S})$. However, all operators considered in this work are symmetric.

Each operator T_j and each base kernel $\mathcal{B}_i$ is represented by their respective name, where operators are the nodes of the tree and base kernels are the leafs. The way how the operators and base kernels are connected is represented through the tree structure. For a given expression tree $\mathcal{T}$, we denote the multiset of all subexpressions/subtrees as $\mathrm{Subtrees}(\mathcal{T})$. Furthermore, we consider paths to the leafs of the tree, for example, for the tree above one path to a leaf would be:

$$\textbf{ADD} \longrightarrow \textbf{MULT} \longrightarrow \textbf{ADD} \longrightarrow \mathrm{PER}$$

We denote the multiset of all paths in the tree $\mathcal{T}$ as $\mathrm{Paths}(\mathcal{T})$. Lastly, we also consider the multiset of all base kernels that exist in a given expression tree as $\mathrm{Base}(\mathcal{T})$. For each described multiset, we denote the number of occurrences of element $\mathcal{E}$ in the multiset as $n(\mathcal{E})$. When building the multisets we also account for two symmetries in the elements that can be applied if an operator is associative and commutative, which we elaborate further in Appendix A.

Depending on the operators that are used, several trees can describe the same kernel $k \in \tilde{\mathbb{K}}$. For technical reasons, we denote with $f : \tilde{\mathbb{K}} \mapsto \mathcal{T}(\tilde{\mathbb{K}})$ a mapping that maps a given kernel $k \in \tilde{\mathbb{K}}$ to one tree $\mathcal{T}$ that induces this kernel, where $\mathcal{T}(\tilde{\mathbb{K}})$ denotes the set of all expression trees that can generate a kernel in $\tilde{\mathbb{K}}$ (see details in Appendix A).

3 Kernel-Kernel

Our kernel-kernel will be defined via a pseudo-metric over the expression trees. Optimal transport (OT) principles have proven themselves to be effective in BO methods over structured spaces, as shown in [5] and [11] for neural architecture search or in [6] for BO over molecule structures. We follow this line of work and also rely on optimal transport, although we only use a simple ground metric to allow for closed-form computations. To allow OT metrics to be used, we summarize each expression tree $\mathcal{T}$ to discrete probability distributions of their building blocks $\mathrm{Base}(\mathcal{T})$, $\mathrm{Paths}(\mathcal{T})$ and $\mathrm{Subtrees}(\mathcal{T})$ via

$$\omega_{\mathrm{base}} := \sum_{\mathcal{E} \in \mathrm{Base}(\mathcal{T})} \omega_{\mathcal{E}} \delta_{\mathcal{E}} \ , \quad \omega_{\mathrm{paths}} := \sum_{\mathcal{E} \in \mathrm{Paths}(\mathcal{T})} \omega_{\mathcal{E}} \delta_{\mathcal{E}} \ , \quad \omega_{\mathrm{subtrees}} := \sum_{\mathcal{E} \in \mathrm{Subtree}(\mathcal{T})} \omega_{\mathcal{E}} \delta_{\mathcal{E}} \ ,$$

where $\delta_{\mathcal{E}}$ is the Dirac delta and $\omega_{\mathcal{E}}$ is the frequency of the element $\mathcal{E}$ in the respective multiset. For example, the frequency of expression $\mathrm{SE} \times \mathrm{PER}$ in the multiset $\mathrm{Subtree}(\mathcal{T})$ is calculated as

$$\omega_{\mathrm{SE} \times \mathrm{PER}} = \frac{n(\mathrm{SE} \times \mathrm{PER})}{|\mathrm{Subtree}(\mathcal{T})|}.$$

Each probability distribution represents a different modeling aspect that is induced by a kernel $k \in \tilde{\mathbb{K}}$ and its corresponding expression tree $\mathcal{T}$:

1. ω_{base} specifies *which* base kernels are present in $\mathcal{T}$, thus, which base assumptions in function space are included such as periodicity, linearity, local smoothness.
2. ω_{paths} specifies *how* the base kernels in $\mathcal{T}$ are used, e.g. whether a periodic component is applied additively or multiplicatively.
3. $\omega_{\mathrm{subtrees}}$ specifies the *interaction* between the base kernels in $\mathcal{T}$, for example if $\mathcal{T}$ contains an addition of a linear and a periodic component or not.

Our pseudo-metric between kernels k_1 and k_2 (and its associated trees $\mathcal{T}_1$ and $\mathcal{T}_2$) uses all three modeling aspects via the optimal transport distances between the respective discrete probability distributions ω_{base}, ω_{paths} and $\omega_{\mathrm{subtrees}}$.

In general the OT distance with ground metric $\tilde{d}$ between two discrete probability distribution $\omega_1 = \sum_{\mathcal{E} \in \Omega} \omega_{1,\mathcal{E}} \delta_{\mathcal{E}}$ and $\omega_2 = \sum_{\mathcal{E} \in \Omega} \omega_{2,\mathcal{E}} \delta_{\mathcal{E}}$ over Ω is defined as

$$W_{\tilde{d}}(\omega_1, \omega_2) = \inf_{\pi \in \mathcal{R}(\omega_1, \omega_2)} \int_{\Omega \times \Omega} \tilde{d}(\mathcal{E}, \mathcal{E}') \pi(d\mathcal{E}, d\mathcal{E}'), \tag{1}$$

where $\pi \in \mathcal{R}(\omega_1, \omega_2)$ is a combined probability distribution over $\Omega \times \Omega$ with marginal distribution $\pi(A \times \Omega) = \omega_1(A)$ and $\pi(\Omega \times B) = \omega_2(B)$ for Borel sets A, B. While for general ground metric

$\tilde{d} : \Omega \times \Omega \to \mathbb{R}$ the optimization problem in (1) can not be solved in closed-form we use as ground metric $\tilde{d}(\mathcal{E}, \mathcal{E}') = \mathbf{1}_{\mathcal{E} \neq \mathcal{E}'}$ which has as closed-form solution the total variation distance between ω_1 and ω_2 (see [19], p. 22), i.e.

$$W_{\tilde{d}}(\omega_1, \omega_2) = \frac{1}{2} \sum_{\mathcal{E} \in \Omega} |\omega_{1,\mathcal{E}} - \omega_{2,\mathcal{E}}|.$$

Utilizing this distance over the modeling assumptions ω_{base}, ω_{paths} and ω_{subtrees} allows for fast computation of the final pseudo-metric and for a proper positive semi definite (p.s.d) kernel-kernel in the end [see Appendix C]. This is not guarantueed for general OT distances (see [11]). We define the final distance between two kernels k_1 and k_2 (and its associated trees $\mathcal{T}_1$ and $\mathcal{T}_2$) as a sum over the OT distances of their respective modeling components

$$\begin{aligned} d(\mathcal{T}_1, \mathcal{T}_2) :=& \alpha_1 W_{\tilde{d}}(\omega_{1,\text{base}}, \omega_{2,\text{base}}) \\ &+ \alpha_2 W_{\tilde{d}}(\omega_{1,\text{paths}}, \omega_{2,\text{paths}}) \\ &+ \alpha_3 W_{\tilde{d}}(\omega_{1,\text{subtrees}}, \omega_{2,\text{subtrees}}), \end{aligned} \tag{2}$$

where $\alpha_i \geq 0$ and $\sum_i \alpha_i = 1$ are weighting parameters that will later be learned automatically via marginal likelihood maximization for GPs.

In case the kernel grammar contains base kernels that act on single dimensions such as SE_i, we define for each dimension $i = 1, \ldots, D$ an individual distribution over base kernels

$$\omega_{\text{base}}^{(i)} := \sum_{\mathcal{E} \in \text{Base}(\mathcal{T}, i)} \omega_{\mathcal{E}} \delta_{\mathcal{E}} \ ,$$

where $\text{Base}(\mathcal{T}, i)$ is the multiset of all base kernels in $\mathcal{T}$ defined on dimension i. We also include the empty-expression $\mathcal{E}_{\text{NULL}}$ for which $\omega_{\mathcal{E}_{\text{NULL}}} = 1$ whenever no base kernel of dimension i is contained in $\mathcal{T}$. $\omega_{\text{base}}^{(i)}$ thus summarizes which base kernels are present that act on dimension i or if the dimension is ignored. The distance in this case is defined as

$$\begin{aligned} d(\mathcal{T}_1, \mathcal{T}_2) :=& \alpha_1 \sum_{i=1}^{D} W_{\tilde{d}}(\omega_{1,\text{base}}^{(i)}, \omega_{2,\text{base}}^{(i)}) \\ &+ \alpha_2 W_{\tilde{d}}(\omega_{1,\text{paths}}, \omega_{2,\text{paths}}) \\ &+ \alpha_3 W_{\tilde{d}}(\omega_{1,\text{subtrees}}, \omega_{2,\text{subtrees}}). \end{aligned} \tag{3}$$

The reason for this distinction is that two kernels k_1 and k_2, which have different active dimensions, are considered to be farther apart with this distance, making it easier to allow variable selection, which is an important aspect of model selection.

Our defined function $d(\mathcal{T}_1, \mathcal{T}_2)$ induces indeed a pseudo-metric in the kernel-grammar generated kernel space, as shown in the following proposition. In particular, it fulfills the triangle inequality.

Proposition 1 *Let $\tilde{\mathbb{K}}$ be the kernel space generated by a kernel grammar. Let $f : \tilde{\mathbb{K}} \mapsto \mathcal{T}(\tilde{\mathbb{K}})$ be a mapping that maps a kernel $k \in \tilde{\mathbb{K}}$ to one of its expression trees $\mathcal{T}$. Then $\hat{d}(k_1, k_2) := d(f(k_1), f(k_2))$ is a pseudo-metric on $\tilde{\mathbb{K}}$ where d is given by (2) or (3) depending on the base kernels in $\tilde{\mathbb{K}}$.*

Given the pseudo-metric, we are now able to define a kernel-kernel with

$$K_{SOT}(k_1, k_2) := \sigma^2 \exp\left(\frac{-\hat{d}(k_1, k_2)}{l^2} \right), \tag{4}$$

which we call *Symbolical-Optimal-Transport (SOT)* kernel-kernel. Here, l denotes the lengthscale and σ^2 the variance of the kernel-kernel. Both parameters are learned in combination with the distance weights via marginal likelihood maximization (see Appendix A).

3.1 Bayesian Optimization for Model Selection

We utilize the proposed kernel-kernel to do model selection for GP's via Bayesian optimization, which is a similar task to [9]. Compared to [9], we use our proposed kernel-kernel, which is a fundamentally different and computationally more efficient way of measuring similarity in GP space.

Table 1: RMSE on test kernel-log-evidence pairs ($\alpha = 0.05$)

Dataset	Hellinger	kNN	Mean	SOT (ours)
Airfoil	0.1773 (0.004)	0.2231 (0.013)	0.3769 (0.002)	**0.0936** (0.003)
Airline	0.3569 (0.007)	0.3813 (0.011)	0.4013 (0.005)	**0.3464** (0.011)
LGBB	0.3772 (0.021)	0.6795 (0.025)	0.8519 (0.016)	**0.2783** (0.008)
Powerplant	0.1912 (0.012)	0.2236 (0.011)	0.2925 (0.009)	**0.0137** (0.005)
Concrete	0.2489 (0.006)	0.1806 (0.010)	0.2912 (0.005)	**0.0451** (0.002)

Given a model selection criteria $g_{\mathcal{D}} : \tilde{\mathbb{K}} \to \mathbb{R}$ for a given dataset $\mathcal{D}$, we want to solve $k^* = \arg\max_{k \in \tilde{\mathbb{K}}} g(k|\mathcal{D})$ via Bayesian optimization. We thus define a surrogate GP model for $g(k|\mathcal{D})$ via

$$f \sim \mathcal{GP}(\mu_c(\cdot), K_{SOT}(\cdot, \cdot)),$$

where $\mu_c(k) = c$ is the constant mean function. As BO acquisition function $a(k|\tilde{\mathcal{D}}_t)$ we use Expected-Improvement (EI). Here, $\tilde{\mathcal{D}}_t$ denotes the set of already queried kernel-selection-criteria pairs $(k, g(k|\mathcal{D}))$ for which the meta-GP posterior $f|\tilde{\mathcal{D}}_t$ is calculated. Starting with an initial set of pairs $\tilde{\mathcal{D}}_0$ we follow the standard BO iterations where in each iteration the acquisition function is maximized $k_t = \arg\max_{k \in \tilde{\mathbb{K}}} a(k|\tilde{\mathcal{D}}_t)$ given the current set of already evaluated kernel-selection-criteria pairs $\tilde{\mathcal{D}}_t$. Then the selection criteria is queried at the chosen kernel $g(k_t|\mathcal{D})$. A complete description of the method can be found in Algorithm 1 in Appendix A.

A crucial part in the BO cycle is the optimization of the acquisition function $\max_{k \in \tilde{\mathbb{K}}} a(k|\tilde{\mathcal{D}}_t)$. While for Euclidean spaces gradient-based methods or grid-based methods could be used to solve that task, this is not an option for a structured space like the considered grammar-generated kernel space. We propose using an evolutionary algorithm to optimize the acquisition function, which can be seen in Algorithm 2 in Appendix A. While an evolutionary algorithm seems to be a computationally intense procedure that takes place in each BO iteration, we emphasize that the evaluation of the acquisition function is very efficient for our proposed method. The reason for the efficiency is that the evaluations of our kernel-kernel $K_{SOT}(k_1, k_2)$ at two kernels $k_1, k_2 \in \tilde{\mathbb{K}}$ is very cheap, compared for example to the method in [9] (see Appendix E for computational time comparision).

3.2 Comparision to Hellinger Kernel-Kernel

The work of [9] also uses BO for kernel selection but with a different principle of measuring the distance in GP space. In particular, they propose measuring the distance of two GP's $\mathcal{M}$ and $\mathcal{M}'$ via the induced prior distributions $p(\mathbf{f}|\mathbf{X}, \mathcal{M})$ in function space evaluated on the design matrix $\mathbf{X}$ of the dataset. Conditioned on the parameters of the GP θ, the distributions $p(\mathbf{f}|\mathbf{X}, \mathcal{M}, \theta)$ are Gaussian and they use the Hellinger distance over Gaussians as base distance between GP models $d(\mathcal{M}_\theta, \mathcal{M}'_{\theta'}|\theta, \theta')$ given the hyperparameters. They finally take the expectation over the hyperparameter priors to construct their final distance. Thus, for each kernel-kernel evaluation $K(\mathcal{M}, \mathcal{M}')$ the integral

$$d(\mathcal{M}, \mathcal{M}') = \int \int d(\mathcal{M}_\theta, \mathcal{M}'_{\theta'}|\theta, \theta') p(\theta) p(\theta') d\theta d\theta'$$

needs to be computed, where the integrand scales cubically in $|\mathbf{X}|$. Each kernel-kernel evaluation is therefore a computationally hard problem by itself. They side-pass this issue partly by selecting a subset of $\mathbf{X}$ as input locations and by using sample-based estimation of the integral. While this could make the method useful for oracles with very long run times, such as model selection for very large data sets, we see conceptual problems in the case of medium oracle run times. For our method, kernel-kernel evaluations $K(k_1, k_2)$ are very efficient as they scale only in the size of the expression tree and no integrals need to be computed.

4 Experiments

In the following section, we show experimental results for our novel meta-GP model and kernel search method. We evaluate our meta-model on a meta-regression task where we predict test kernel-log-evidence pairs based on training pairs. Secondly, we consider kernel search and compare it

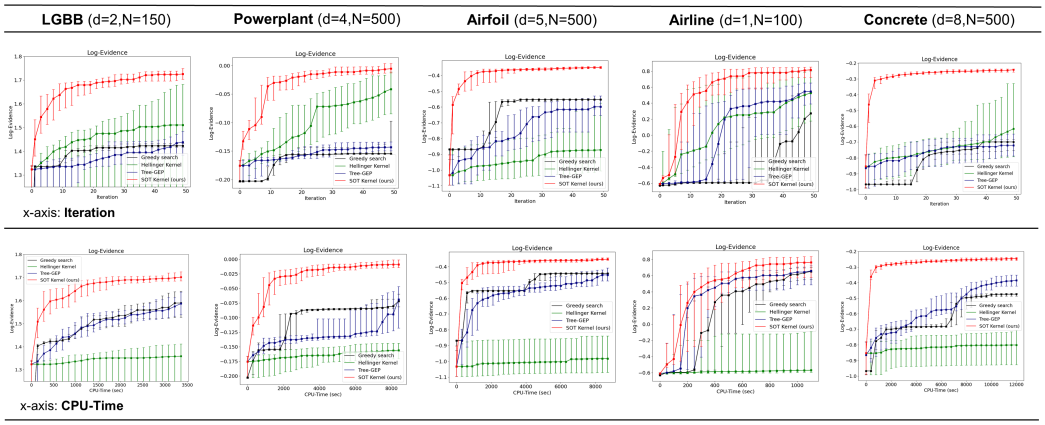


Figure 1: Plots for the model selection task over number of evaluated models (top) and CPU-time (bottom) showing the normalized log-evidence of the best model found up to this iteration/time point.

against the greedy method in [3], the evolutionary algorithm in [4] and against the BO method of [9]. The implementation of our method is available at `https://github.com/boschresearch/bosot`.

Selection Criteria and Datasets: In all our experiments, we consider the normalized log-model evidence $g(k|\mathcal{D}) = \log p(\mathbf{y}|\mathbf{X}, k)/|\mathcal{D}|$ as model selection criteria, as also done in [9]. We always use the Laplace approximation to calculate the log-model evidence, where we use 10 repeats to do MAP estimation of the kernel parameters. Furthermore, we consider the following publicly available datasets: Airline, LGBB, Airfoil, Powerplant, Concrete. Airline is a one dimensional time-series dataset, LGBB is a two-dimensional dataset with low observation noise. Powerplant, Airfoil and Concrete are four, five, and eight dimensional, whereas Powerplant exhibits higher observation noise. We use medium-sized training sets - 100 and 150 datapoints for Airline and LGBB and 500 datapoints for the other three datasets. All training sets are uniformly drawn from the full datasets. The outputs are normalized and the inputs are scaled to be in the unit interval. Further details can be found in Appendix D.

Search Spaces: We consider two search spaces. The first consists of the base kernels $\mathrm{SE}_i, \mathrm{LIN}_i, \mathrm{PER}_i, \mathrm{RQ}_i$ and the operators $+$ and $\times$. This is the space considered in [3] for time series and low-dimensional datasets. The second search space uses as base kernels SE_i and RQ_i and also $+$ and $\times$ as operators and was considered in [3] for higher dimensional base datasets. We consider the first space for Airline and LGBB and the second for Powerplant, Airfoil, and Concrete. The hyperparameter priors for the base kernels can be found in the Appendix A.

Prediction of Selection Criteria: We evaluate our meta-GP model on a meta-regression task by predicting test kernel-log-evidence pairs based on training pairs. The quality of predictions in kernel space might also be an indicator for good performance in BO for kernel search. Given the base dataset $\mathcal{D}$ we create a training set $\tilde{\mathcal{D}}_{\text{train}} = \{(k_i, g(k_i|\mathcal{D}))|k_i \in \tilde{\mathbb{K}}_{\text{train}} \subset \tilde{\mathbb{K}}, i = 1, \dots, n_{\text{train}}\}$ and a test set $\tilde{\mathcal{D}}_{\text{test}} = \{(k_i, g(k_i|\mathcal{D}))|k_i \in \tilde{\mathbb{K}}_{\text{test}} \subset \tilde{\mathbb{K}}, i = 1, \dots, n_{\text{test}}\}$ - each containing 500 kernel-log-evidence pairs. We generate the train and tests sets exactly as in [9], where we create one set $\tilde{\mathbb{K}}_{\text{complete}}$ by first initializing it with all base kernels and iteratively pick one kernel of the current set, apply one random operation of the kernel grammar and add the resulting kernel to the current set and repeat this process until we have $n_{\text{train}} + n_{\text{test}}$ kernels in $\tilde{\mathbb{K}}_{\text{complete}}$. We then divide the set uniformly into $\tilde{\mathbb{K}}_{\text{train}}$ and $\tilde{\mathbb{K}}_{\text{test}}$. We compare our model to the mean-predictor as baseline, which just predicts the mean of the train set at each test point and a kNN predictor based on the kernel-grammar operations [see Appendix D or [9]]. Furthermore, we compare against the meta-GP model of [9]. We report root mean squared error (RMSE) scores on the test sets in Table 1. It can be observed that our method leads to more precise predictions on all four meta prediction tasks, indicating that the symbolical representations already contain sufficient information to predict log-evidence values.

Table 2: Predictive negative log-likelihood values on test set at final time stamp. Values are marked bold if they are not significantly different from the best value according to a two-sample t-test ($\alpha = 0.05$).

Dataset	Greedy	Hellinger	Tree-GEP	SOT (ours)
Airline	-0.4042 (0.615)	0.3368 (0.207)	**-0.5594** (0.580)	**-0.7015** (0.471)
LGBB	-0.7528 (0.661)	**-0.8787** (0.854)	**-1.0701** (0.532)	**-0.9325** (0.492)
Powerplant	-0.0053 (0.054)	0.0580 (0.037)	-0.0241 (0.057)	**-0.0661** (0.032)
Airfoil	**0.0837** (0.026)	0.9080 (0.205)	0.1826 (0.118)	**0.1006** (0.090)
Concrete	0.3254 (0.019)	0.6633 (0.207)	**0.2812** (0.074)	**0.2872** (0.044)

Model Selection - Setup: Concerning kernel search, we compare against the BO method that employs the Hellinger kernel-kernel [9], against greedy search [3] and against the evolutionary algorithm presented in [4], referred to as TreeGEP. Both BO methods run for 50 iterations and the kernel-kernel hyperparameters are updated in each iteration via marginal likelihood maximization. Our method applied the evolutionary Algorithm 2 (see Appendix A) to optimize its acquisition function, using a population size of 100. As it is computationally unfeasible to apply the same kind of acquisition function optimization for the Hellinger kernel-kernel we use their method of optimizing the acquisition function where an active set of kernels is updated in each iteration. Both BO methods and TreeGEP start with the same set of initial kernels, for which we apply two random grammar operations for each base kernel. Greedy search by design needs to start from the empty kernel. We give it a head start by the number of initial datapoints (see Appendix D). For each dataset, we display results from 30 independent runs with different seeds, namely medians and quartiles of the log-evidence score over iterations and CPU-time in Figure 1. The implementation of both BO methods is based on *GPflow* [10]. Further experimental details and parameter settings for all methods can be found in Appendix D.

Model Selection - Results: As shown in Figure 1, we outperform all methods in terms of performance over number of model evaluations. This is not surprising against the two heuristics, as they are not optimized towards keeping the number of model evaluations low. However, it is notable that we are more sample-efficient compared to [9], who solve a much harder problem for computing their kernel-kernel. In terms of performance over CPU-time, we outperform all methods on $\mathrm{Airfoil}, \mathrm{Powerplant}, \mathrm{LGBB}$ and $\mathrm{Concrete}$ significantly. On $\mathrm{Airline}$, the advantage over the heuristics is smaller - the reason is the lower oracle time - which benefits the heuristics that do not need to optimize the acquisition function. The reason for the poor performance of the Hellinger kernel-kernel in terms of CPU-time is the high ratio of acquisition function optimization to oracle time, which was as high as $50:1$ in our experiments (see detailed numbers in Appendix D).

Test Performance: When optimizing a model selection criteria, one expects that this also materializes in a better test performance. In Table 2 we therefore show the predictive negative log-likelihood (NLL) scores on a held out test-set of the selected models at the final time stamp. We observe that the advantage in the model selection value gets transferred to the test performance.

Further Experiments: In Appendix E we include further investigations on the behavior of our method on simulated data coming from a ground-truth kernel. Furthermore, we include a comparison of the selected kernel to the standard RBF kernel and against Functional Kernel Learning (FKL) [2].

5 Limitations

In case the dataset size is very small, greedy search or evolutionary algorithms might have an advantage in terms of computational time as the optimization of the acquisition function outweighs the computation of the model selection criteria. Thus, in these instances our method might not show a strong benefit. Furthermore, the kernel grammar is often used for downstream applications that use the selected hypothesis for interpretation, such as [7] who build an automatic natural language description of the selected hypothesis. Depending on the dataset and how many steps are employed in the acquisition function optimization, relatively large hypotheses might be found as optimal. This

might render the interpretation difficult. However, we show in Appendix E example configurations of our algorithm that can be used to get smaller, well interpretable hypothesis.

6 Conclusion

We presented a novel way of doing BO for model selection of GP's by measuring the distance between two GP's via the symbolical description of the underlying statistical hypothesis. The main contribution is the deduced pseudo-metric over kernels and the resulting SOT kernel-kernel. We show that our approach leads to a more efficient way of searching through a discrete kernel space compared to other BO methods and search heuristics.

References

[1] F. Bach. Exploring large feature spaces with hierarchical multiple kernel learning. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems*, volume 21. Curran Associates, Inc., 2008.

[2] G. W. Benton, W. J. Maddox, J. P. Salkey, J. Albinati, and A. G. Wilson. *Function-Space Distributions over Kernels*. Curran Associates Inc., Red Hook, NY, USA, 2019.

[3] D. Duvenaud, J. Lloyd, R. Grosse, J. Tenenbaum, and G. Zoubin. Structure discovery in nonparametric regression through compositional kernel search. In S. Dasgupta and D. McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1166–1174, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

[4] S.-S. Jin. Compositional kernel learning using tree-based genetic programming for gaussian process regression. *Struct. Multidiscip. Optim.*, 62(3):1313–1351, sep 2020.

[5] K. Kandasamy, W. Neiswanger, J. Schneider, B. Póczos, and E. P. Xing. Neural architecture search with bayesian optimisation and optimal transport. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 2020–2029, Red Hook, NY, USA, 2018. Curran Associates Inc.

[6] K. Korovina, S. Xu, K. Kandasamy, W. Neiswanger, B. Poczos, J. Schneider, and E. Xing. Chembo: Bayesian optimization of small organic molecules with synthesizable recommendations. In S. Chiappa and R. Calandra, editors, *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pages 3393–3403. PMLR, 26–28 Aug 2020.

[7] J. R. Lloyd, D. Duvenaud, R. Grosse, J. B. Tenenbaum, and Z. Ghahramani. Automatic construction and natural-language description of nonparametric regression models. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, AAAI'14, page 1242–1250. AAAI Press, 2014.

[8] D. J. C. MacKay. Bayesian interpolation. *Neural Computation*, 4(3):415–447, 05 1992.

[9] G. Malkomes, C. Schaff, and R. Garnett. Bayesian optimization for automated model selection. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[10] A. G. d. G. Matthews, M. van der Wilk, T. Nickson, K. Fujii, A. Boukouvalas, P. León-Villagrá, Z. Ghahramani, and J. Hensman. GPflow: A Gaussian process library using TensorFlow. *Journal of Machine Learning Research*, 18(40):1–6, apr 2017.

[11] V. Nguyen, T. Le, M. Yamada, and M. A. Osborne. Optimal transport kernels for sequential and parallel neural architecture search. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 8084–8095. PMLR, 18–24 Jul 2021.

[12] S. W. Ober, C. E. Rasmussen, and M. van der Wilk. The promises and pitfalls of deep kernel learning. In C. de Campos and M. H. Maathuis, editors, *Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence*, volume 161 of *Proceedings of Machine Learning Research*, pages 1206–1216. PMLR, 27–30 Jul 2021.

[13] Rasmussen and Williams. *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, MA, USA, Jan. 2006.

[14] C. Rasmussen and Z. Ghahramani. Occam's razor. In T. Leen, T. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems*, volume 13. MIT Press, 2000.

[15] J. Schreiter, D. Nguyen-Tuong, M. Eberts, B. Bischoff, H. Markert, and M. Toussaint. Safe exploration for active learning with gaussian processes. In *Proceedings of the 2015th European Conference on Machine Learning and Knowledge Discovery in Databases - Volume Part III*, ECMLPKDD'15, page 133–149, Gewerbestrasse 11 CH-6330, Cham (ZG), CHE, 2015. Springer.

[16] J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

[17] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In F. Pereira, C. Burges, L. Bottou, and K. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[18] S. Sun, G. Zhang, C. Wang, W. Zeng, J. Li, and R. Grosse. Differentiable compositional kernel learning for Gaussian processes. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4828–4837. PMLR, 10–15 Jul 2018.

[19] C. Villani. *Optimal Transport: Old and New*. Grundlehren der mathematischen Wissenschaften. Springer Berlin Heidelberg, 2008.

[20] A. Wilson and R. Adams. Gaussian process kernels for pattern discovery and extrapolation. In S. Dasgupta and D. McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1067–1075, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

[21] A. G. Wilson, E. Gilboa, A. Nehorai, and J. P. Cunningham. Fast kernel learning for multidimensional pattern extrapolation. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[22] A. G. Wilson, Z. Hu, R. Salakhutdinov, and E. P. Xing. Deep kernel learning. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, volume 51 of *Proceedings of Machine Learning Research*, pages 370–378, Cadiz, Spain, 2016. PMLR.

[23] C. Zimmer, M. Meister, and D. Nguyen-Tuong. Safe active learning for time-series modeling with gaussian processes. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] We clarify our main contributions in the end of the introduction and provide the mentioned method in Section 3 and the mentioned experimental results in Section 4.
   (b) Did you describe the limitations of your work? [Yes] In Section 5
   (c) Did you discuss any potential negative societal impacts of your work? [No] No obvious direct negative societal impacts.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] Given in Definition 1, Proposition 1 and in Section C
   (b) Did you include complete proofs of all theoretical results? [Yes] In Section C
3. If you ran experiments...

(a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [N/A] We will provide code as soon as clearance of code is completed.

(b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] Either in Section 4 or Appendix D

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] We report either standard deviations or quartiles.

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] We analyze the search performance over CPU-time.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes] Yes we cite the GPflow related work, which is the main python framework we are using.

(b) Did you mention the license of the assets? [No]

(c) Did you include any new assets either in the supplemental material or as a URL? [Yes] Code is available at `https://github.com/boschresearch/bosot`.

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

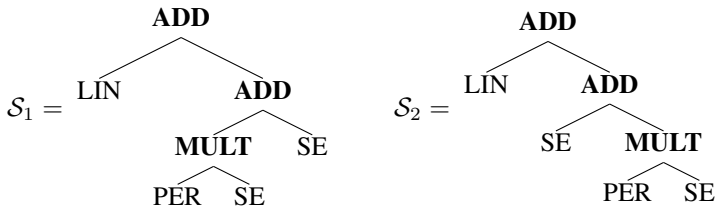
(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Method Details

In the following section, we specify further details of the proposed method. We show the symmetries we employ in the multisets and give details on the approximation of the log-evidence. Furthermore, we describe the base kernels and their parameter priors, the marginal likelihood maximization of the kernel-kernel hyperparameters and how the acquisition function optimization is done.

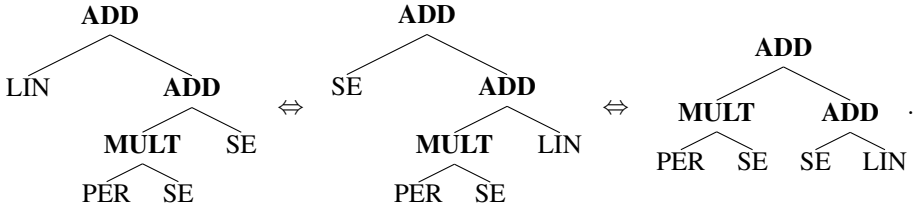
A.1 Symmetries in the Multisets

Our method can be used with general operators $T : \mathcal{K} \times \mathcal{K} \to \mathcal{K}$ in kernel space. However, depending on the concrete operators, one might also incorporate properties that leave the expression/subexpressions unchanged. In particular, addition and multiplication are commutative and associative. For commutative operators two trees/subtrees describe the same kernel if one rotates the nodes under a commutative operator, e.g.



This symmetry is automatically considered in the multisets $\mathrm{Base}(\mathcal{T})$ and $\mathrm{Path}(\mathcal{T})$ as the base kernels and paths stay unchanged when rotating two nodes. For $\mathrm{Subtree}(\mathcal{T})$, we account explicitly for that symmetry via hashing functions that are invariant to rotation of subtrees. This allows counting tree $\mathcal{S}_1$ and tree $\mathcal{S}_2$ as an identical subtree $\tilde{\mathcal{S}}$ in the multiset $\mathrm{Subtree}(\mathcal{T})$.

For the multiset $\mathrm{Path}(\mathcal{T})$, we furthermore consider a symmetry that exists in case an operator is associative and commutative. In this case, one can exchange the nodes of two consecutive operators of the same kind without changing the expression, e.g.



We incorporate this symmetry into $\mathrm{Path}(\mathcal{T})$ by only considering the smallest path to that base kernel that exist in an equivalent expression (equivalent under this symmetry). This is realized in $\mathrm{Path}(\mathcal{T})$ by counting identical operators in a row along a path only as one, e.g. the following paths would be considered the same:

$$\mathbf{ADD} \longrightarrow \mathbf{ADD} \longrightarrow \mathbf{MULT} \longrightarrow \mathrm{PER}\,,$$
$$\mathbf{ADD} \longrightarrow \mathbf{MULT} \longrightarrow \mathrm{PER}\,.$$

While one might also integrate more symmetries that stem from, e.g. the distributive property of the multiplication, we found that these two symmetries are particularly easy and efficient to implement. The symmetries are also not restricted to multiplication or addition. The first symmetry can be used whenever T is associative, thus, $T(k_1, k_2) = T(k_2, k_1)$ and the second whenever T is associative and commutative, thus, $T(k_1, T(k_2, k_3)) = T(k_2, T(k_1, k_3)) = T(T(k_1, k_3), k_2)$. The motivation of considering symmetries at all is that two trees $\mathcal{T}_1$ and $\mathcal{T}_2$ are considered more similar in case they share the same structure (with respect to the symmetry) and, therefore, similar kernels can be detected more easily.

A.2 Mapping from Kernels to Trees

As we already observed in the previous section, two different trees $\mathcal{T}_1$ and $\mathcal{T}_2$ might describe the same kernel k. For technical reasons, we therefore consider in Proposition 1 a mapping $f : \mathbb{K} \to \mathcal{T}(\tilde{\mathbb{K}})$

that maps a kernel always to the same tree. Concretely, this is done such that the pseudo metric is defined in kernel space rather than in tree space. When using addition and multiplication, one could check if $\mathcal{T}_1$ and $\mathcal{T}_2$ describe the same expression via recursive hashes that follow the same rules as addition and multiplication. One could use this check to implement such a function $f : \mathbb{K} \to \mathcal{T}(\tilde{\mathbb{K}})$. However, in our experiments we directly deal with the trees (the BO and the acquisition function algorithm directly act on the trees anyway) and ignore filtering out potential collisions of two trees - we did not observe any downsides of doing that.

A.3 Approximation of Log-Model-Evidence

In all our experiments, we use the normalized log-model-evidence $g(k|\mathcal{D}) = \log p(\mathbf{y}|\mathbf{X}, k)/|\mathcal{D}|$ as selection criteria. Similar to [9] we use the Laplace approximation to approximate the log-evidence, which is (see [9])

$$\log p(\mathbf{y}|\mathbf{X}, k) \approx \log p(\mathbf{y}|\mathbf{X}, k, \hat{\gamma}) + \log p(\hat{\gamma}) - \frac{1}{2}\log \det \Sigma^{-1} + \frac{d}{2} \log 2\pi,$$

where $\gamma = (\theta, \sigma^2) \in \mathbb{R}^d$ are the parameters of the kernel and the likelihood variance, $d \in \mathbb{N}$ is the number of parameters, $\hat{\gamma}$ denotes the MAP estimate of γ, and $\Sigma^{-1} = -\nabla^2 \log p(\gamma|\mathcal{D}, k)|_{\gamma=\hat{\gamma}}$. Creating the MAP estimate of γ scales cubically in the dataset size $|\mathcal{D}|$ in each optimization step. We use the LBFGS optimizer to create the MAP estimate. As the loss function is non-convex, we make 10 restarts with random initialization of the initial parameters.

A.4 Base Kernels and Priors on Parameters:

Here, we specify the base kernels that are used, including their priors on the parameters. We chose the parameter priors such that broad priors in function space are induced (Here, we assume that the datasets contain normalized outputs and inputs scaled to the unit interval). All base kernels are defined on $\mathbb{R}$ and are applied on dimension i if this is indicated by the base kernel symbol, e.g. SE_i. We consider the following base kernels:

1. Squared Exponential SE:

$$k(x, x') = \sigma^2 \exp\left(-\frac{1}{2} \frac{(x - x')^2}{l^2} \right)$$

   with $l \sim \mathrm{Gamma}(2.0, 2.0)$ and $\sigma^2 \sim \mathrm{Gamma}(2.0, 3.0)$,

2. Periodic PER:

$$k(x, x') = \sigma^2 \exp\left(-\frac{1}{2} \frac{\sin^2(\pi|x - x'|/p)}{l^2} \right)$$

   with $l \sim \mathrm{Gamma}(2.0, 2.0)$, $\sigma^2 \sim \mathrm{Gamma}(2.0, 3.0)$, and $p \sim \mathrm{Gamma}(2.0, 2.0)$,

3. Linear LIN:

$$k(x, x') = \sigma^2 x\, y + \sigma_c^2$$

   with $\sigma^2 \sim \mathrm{Gamma}(2.0, 3.0)$ and $\sigma_c^2 \sim \mathrm{Gamma}(2.0, 3.0)$,

4. Rational Quadratic RQ:

$$k(x, x') = \sigma^2 \left(1 + \frac{(x - x')^2}{2\alpha l^2} \right)^{-\alpha}$$

   with $l \sim \mathrm{Gamma}(2.0, 2.0)$, $\sigma^2 \sim \mathrm{Gamma}(2.0, 3.0)$, and $\alpha \sim \mathrm{Gamma}(2.0, 2.0)$.

A.5 Marginal Likelihood Maximization of Kernel-Kernel Parameters

Doing GP regression with the meta GP model $f \sim \mathcal{GP}(\mu_c(\cdot), K_{SOT}(\cdot, \cdot))$ involves fitting some parameters, namely, the constant $c \in \mathbb{R}$ in the mean function, the kernel-kernel variance σ^2, the kernel-kernel lengthscale l^2, the distance weights $\alpha_1, \alpha_2, \alpha_3$, and the likelihood variance denoted

Function `BOKernelSearch`($\mathcal{D}$,T,L,$\mathrm{n}_{\mathrm{initial}}$)**:**

$\tilde{\mathcal{D}}_0 = \mathrm{GetInitialDataset}(\mathcal{D}, \mathrm{n}_{\mathrm{initial}})$

for $t = 0, \ldots, T-1$ **do**

Fit Meta GP model $f \sim \mathcal{GP}(\mu_c(\cdot), K_{SOT}(\cdot,\cdot))$ to $\tilde{\mathcal{D}}_t$

$k_t \leftarrow$ `EvolutionaryAlg`($a(\cdot|\tilde{\mathcal{D}}_t)$,$L$)

Query model selection criteria $g_t \leftarrow g(k_t|\mathcal{D})$

$\tilde{\mathcal{D}}_{t+1} = \tilde{\mathcal{D}}_t \cup \{(k_t, g_t)\}$

end

$t^* \leftarrow \arg\max_{t=0,\ldots,T-1} g_t$

return k_{t^*}

Algorithm 1: BO for kernel search via SOT kernel-kernels.

by σ_g^2. The distance weights are reparameterized with $\alpha_i := \frac{\sigma(\tilde{\alpha}_i)}{\sum_{j=1}^{3}\sigma(\tilde{\alpha}_j)}, \tilde{\alpha}_j \in \mathbb{R}$, where $\sigma(\cdot)$ is the standard sigmoid function. Thus, $\alpha_1 + \alpha_2 + \alpha_3 = 1$ and $\alpha_i \geq 0$. For the other parameters, we use standard *GPflow* bijectors to transform them to their domain of definition. Let θ denote all kernel-kernel parameters and let $\tilde{\mathcal{D}} = \{(k_j, g(k_j|\mathcal{D}))|j = 1, \ldots, R\}$ denote the kernel-selection-criteria pairs to which the meta-GP model is to be fitted. We fit the parameter via maximization of the log-marginal likelihood

$$(\theta^*, \sigma_g^*, c^*) = \arg\max_{\theta,\sigma_g,c} \log \mathcal{N}(\mathbf{g}; \mu_c(\mathbf{X}_{\mathrm{kernels}}), K_{SOT,\theta}(\mathbf{X}_{\mathrm{kernels}}, \mathbf{X}_{\mathrm{kernels}}) + \sigma_g^2\mathbf{I}) \tag{5}$$

with $\mathbf{g} = [g(k_1|\mathcal{D}), \ldots, g(k_R|\mathcal{D})]^\intercal$ and $\mathbf{X}_{\mathrm{kernels}} = \{k_1, \ldots, k_R\}$.

A.6 BO Algorithm and Acquisition Function Optimization

In Algorithm 1, we show the BO steps for kernel search. First, we draw an initial dataset of kernel-selection-criteria pairs. Our base setting applies two random grammar operations from each base kernel. The meta-GP parameters are fitted in each BO optimization. The acquisition function is optimized via the evolutionary algorithm in 2. This algorithm searches in the hypotheses space from small to big hypotheses. Given an initial population of kernels, we calculate the acquisition function on all kernels in the population. Then the n_{survive} best kernels in the population survive. Each selected kernel gets $n_{\mathrm{offspring}}$ children by generating $n_{\mathrm{offspring}}$ new kernels via applying one random grammar operation. The new population is formed via the selected kernel combined with its offspring. L is the number of iterations in the evolutionary algorithm. In combination with the initial population it determines how many base kernels the hypotheses can contain maximally. In each iteration, the number of base kernels in the best hypothesis can maximally grow by one.

Function `EvolutionaryAlg`($a(\cdot|\tilde{\mathcal{D}}_t)$,$L$)**:**

$\mathrm{n}_{\mathrm{survive}} = \frac{\mathrm{n}_{\mathrm{population}}}{(\mathrm{n}_{\mathrm{offspring}}+1)}$

$\mathcal{K}_0 = \mathrm{GetInitialKernels}(\mathrm{n}_{\mathrm{population}})$

for $l = 0, \ldots, L-1$ **do**

$\mathrm{fitness}_l = a(\mathcal{K}_l|\tilde{\mathcal{D}}_t)$

$\mathcal{K}_l^{\mathrm{selected}} = \mathrm{Select}(\mathcal{K}_l, \mathrm{fitness}_l, \mathrm{n}_{\mathrm{survive}})$

$\mathcal{K}_l^{\mathrm{offspring}} = \emptyset$

for k **in** $\mathcal{K}_l^{\mathrm{selected}}$ **do**

$\mathcal{K}_l^{\mathrm{offspring}} \leftarrow \mathcal{K}_l^{\mathrm{offspring}} \cup \mathrm{ApplyGrammarOps}(k, \mathrm{n}_{\mathrm{offspring}})$

end

$\mathcal{K}_{l+1} = \mathcal{K}_l^{\mathrm{selected}} \cup \mathcal{K}_l^{\mathrm{offspring}}$

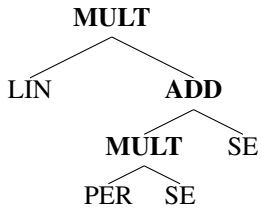
end

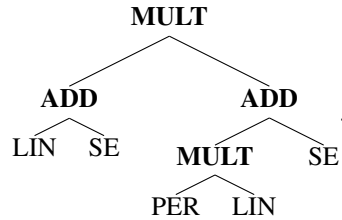
return $\arg\max(a(\mathcal{K}_L|\tilde{\mathcal{D}}_t))$

Algorithm 2: Evolutionary algorithm for BO kernel search.

B Example Calculation of SOT Kernel-Kernel

In this section, we give a small example calculation of our proposed kernel-kernel. We calculate our proposed pseudo-metric $d_{SOT}(k_1, k_2)$ for the two kernels k_1 and k_2 with the following expression trees

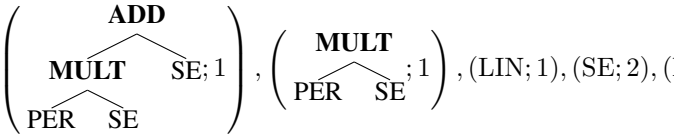
$\mathcal{T}_1 =$ 

and $\mathcal{T}_2 =$ 

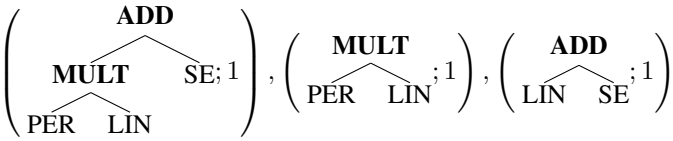
.

In the first step, we extract the tree features and create the multisets. Here, we use the notation $\{(\mathcal{E}_1; n(\mathcal{E}_1)), (\mathcal{E}_2; n(\mathcal{E}_2)), \dots\}$ to denote the existence of an element $\mathcal{E}$ in the multiset as well as the cardinality of the element. We obtain the following multisets:

$$
\begin{aligned}
&\mathrm{Base}(\mathcal{T}_1) = \{(\mathrm{LIN}; 1), (\mathrm{SE}; 2), (\mathrm{PER}; 1)\},\\
&\mathrm{Base}(\mathcal{T}_2) = \{(\mathrm{LIN}; 2), (\mathrm{SE}; 2), (\mathrm{PER}; 1)\},\\
&\mathrm{Path}(\mathcal{T}_1) = \{(\mathbf{MULT} \longrightarrow \mathbf{ADD} \longrightarrow \mathbf{MULT} \longrightarrow \mathrm{PER}; 1),\\
&\qquad (\mathbf{MULT} \longrightarrow \mathbf{ADD} \longrightarrow \mathbf{MULT} \longrightarrow \mathrm{SE}; 1),\\
&\qquad (\mathbf{MULT} \longrightarrow \mathbf{ADD} \longrightarrow \mathrm{SE}; 1)\\
&\qquad (\mathbf{MULT} \longrightarrow \mathrm{LIN}; 1)\},\\
&\mathrm{Path}(\mathcal{T}_2) = \{(\mathbf{MULT} \longrightarrow \mathbf{ADD} \longrightarrow \mathbf{MULT} \longrightarrow \mathrm{PER}; 1),\\
&\qquad (\mathbf{MULT} \longrightarrow \mathbf{ADD} \longrightarrow \mathbf{MULT} \longrightarrow \mathrm{LIN}; 1),\\
&\qquad (\mathbf{MULT} \longrightarrow \mathbf{ADD} \longrightarrow \mathrm{SE}; 2)\\
&\qquad (\mathbf{MULT} \longrightarrow \mathbf{ADD} \longrightarrow \mathrm{LIN}; 1)\},
\end{aligned}
$$

$\mathrm{Subtree}(\mathcal{T}_1) = \{(\mathcal{T}_1; 1),$ 

$, (\mathrm{LIN}; 1), (\mathrm{SE}; 2), (\mathrm{PER}; 1)\},$

$\mathrm{Subtree}(\mathcal{T}_2) = \{(\mathcal{T}_2; 1),$ 

$, (\mathrm{LIN}; 2),$
$(\mathrm{SE}; 2), (\mathrm{PER}; 1)\}.$

Next, we build the probability vector for each feature multiset, for example, for the base kernels:

$$
\begin{aligned}
\omega_{1,\mathrm{base}} &= \frac{1}{4}\delta_{\mathrm{LIN}} + \frac{1}{2}\delta_{\mathrm{SE}} + \frac{1}{4}\delta_{\mathrm{PER}}\,,\\
\omega_{1,\mathrm{base}} &= \frac{2}{5}\delta_{\mathrm{LIN}} + \frac{2}{5}\delta_{\mathrm{SE}} + \frac{1}{5}\delta_{\mathrm{PER}}\,,
\end{aligned}
$$

and calculate the total variation distance between $\omega_{1,\mathrm{base}}$ and $\omega_{2,\mathrm{base}}$:

$$
W_{\tilde{d}}(\omega_{1,\mathrm{base}}, \omega_{2,\mathrm{base}}) = \frac{1}{2}\left(\left|\frac{1}{4} - \frac{2}{5}\right| + \left|\frac{1}{2} - \frac{2}{5}\right| + \left|\frac{1}{4} - \frac{1}{5}\right|\right) = \frac{3}{20}.
$$

For the paths, the total variation distance results to:

$$
W_{\tilde{d}}(\omega_{1,\mathrm{path}}, \omega_{2,\mathrm{path}}) = \frac{1}{2}\left(\left|\frac{1}{4} - \frac{1}{5}\right| + \left|\frac{1}{4} - 0\right| + \left|\frac{1}{4} - \frac{2}{5}\right| + \left|\frac{1}{4} - 0\right| + \left|0 - \frac{1}{5}\right| + \left|0 - \frac{1}{5}\right|\right) = \frac{11}{20}.
$$

For the subtrees, the total variation distance results to:

$$
\begin{aligned}
W_{\tilde{d}}(\omega_{1,\mathrm{path}}, \omega_{2,\mathrm{path}}) = \frac{1}{2}\Bigg(&\left|\frac{1}{7} - 0\right| + \left|\frac{1}{7} - 0\right| + \left|\frac{1}{7} - 0\right| + \left|\frac{1}{7} - \frac{2}{9}\right| + \left|\frac{2}{7} - \frac{2}{9}\right| + \left|\frac{1}{7} - \frac{1}{9}\right|\\
&+ \left|0 - \frac{1}{9}\right| + \left|0 - \frac{1}{9}\right| + \left|0 - \frac{1}{9}\right| + \left|0 - \frac{1}{9}\right|\Bigg) = \frac{11}{21}.
\end{aligned}
$$

Function `GetKernel(`$\mathcal{T}$`)`**:**
  if *Root of* $\mathcal{T}$ *is leaf* **then**
    return base kernel $\mathcal{B}$ associated with leaf
  else
    return $T(\texttt{GetKernel}(\mathcal{T}_L), \texttt{GetKernel}(\mathcal{T}_R))$ where T is the operator associated with the root of $\mathcal{T}$ and $\mathcal{T}_L$ and $\mathcal{T}_R$ are the left and right subtree below the root of $\mathcal{T}$.
  end

Algorithm 3: Get a kernel from an expression tree.

The complete distance given the weights is then

$$d(k_1, k_2) = \alpha_1 \frac{3}{20} + \alpha_2 \frac{11}{20} + \alpha_3 \frac{11}{21}$$

and

$$K_{SOT}(k_1, k_2) := \sigma^2 \exp\left(\frac{-\tilde{d}(k_1, k_2)}{l^2}\right).$$

The parameters are learned via marginal likelihood maximization and are dependent on the dataset.

C Technical Details and Proofs

When referring to a kernel k from the kernel-grammar generated kernel space $\tilde{\mathbb{K}}$ such as writing $k \in \tilde{\mathbb{K}}$, we actually refer to the associated kernel family over its parameters $\{k_\theta | \theta \in \Theta\}$. Applying an operator T onto two kernels k_1 and k_2 then results in another kernel family $\{T(k_{1,\theta_1}, k_{2,\theta_2}) | \theta_1 \in \Theta_1, \theta_2 \in \Theta_2\}$ and we refer with the notation $T(k_1, k_2)$ to this kernel family with new parameter space $\Theta_1 \times \Theta_2$. In case the same kernel family $\{\tilde{k}_\theta | \theta \in \Theta\}$ appears twice in an operator the parameters are not shared, e.g. $T(\tilde{k}, \tilde{k})$ corresponds to the family $\{T(\tilde{k}_{\theta_1}, \tilde{k}_{\theta_2}) | \theta_1 \in \Theta, \theta_2 \in \Theta\}$ with new parameter space $\Theta \times \Theta$.

Definition 2 *We call a binary tree $\mathcal{T}$ whose nodes are associated with the operators $\{T_1, \ldots, T_l\}$ and whose leafs are associated with the base kernels $\{\mathcal{B}_1, \ldots, \mathcal{B}_r\}$ an expression tree of a kernel k if k is constructed by applying the operators recursively onto the leafs, meaning k is the result of Algorithm 3.*

Any kernel k in a kernel-grammar generated kernel space has an expression tree $\mathcal{T}$ via the construction of $\tilde{\mathbb{K}}$. We further denote the set of all trees that can generate a kernel in $\tilde{\mathbb{K}}$ as $\mathcal{T}(\tilde{\mathbb{K}}) := \{\mathcal{T} | \exists k \in \tilde{\mathbb{K}} : k \text{ is result of } \mathrm{GetKernel}(\mathcal{T})\}$.

Proof of Proposition 1: W.l.o.g. we consider $\tilde{\mathbb{K}}$ without separate base kernels for dimensions. We denote by $g_i : \mathcal{T}(\tilde{\mathbb{K}}) \to [0, 1]^{L_i}$ and $i \in \{\text{base}, \text{path}, \text{subtree}\}$ the mappings from expression trees to the probability vectors $\omega_{\text{base}}, \omega_{\text{paths}}, \omega_{\text{subtrees}}$, where L_i denotes the number of different elements of the respective type (e.g. number of different paths to the leafs for expression trees of depth M). Then

$$\begin{aligned}\hat{d}(k_1, k_2) = d(f(k_1), f(k_2)) = &\frac{\alpha_1}{2} \|g_1(f(k_1)) - g_1(f(k_2))\|_1 \\ &+ \frac{\alpha_2}{2} \|g_2(f(k_1)) - g_2(f(k_2))\|_1 \\ &+ \frac{\alpha_3}{2} \|g_3(f(k_1)) - g_3(f(k_2))\|_1\end{aligned}$$

is a pseudo metric as chaining of a metric with a mapping results in a pseudo metric, and the positive-weighted sum of pseudo metrics still is a pseudo metric. ■

Proposition 2 *Let $\tilde{\mathbb{K}}$ be the kernel space generated by a kernel grammar. Then*

$$K_{SOT}(k_1, k_2) := \sigma^2 \exp\left(\frac{-\hat{d}(k_1, k_2)}{l^2}\right) \tag{6}$$

is a proper p.s.d. kernel over $\tilde{\mathbb{K}}$.

Proof: As in the proof of Proposition 1 we can write $\hat{d}(k_1, k_2)$ as a weighted sum of Manhatten metrics, which leads to:

$$K_{SOT}(k_1, k_2) := \sigma^2 \exp\left(\frac{-\hat{d}(k_1, k_2)}{l^2}\right)$$
$$= \sigma^2 \prod_{i=1}^{3} \exp(-\frac{\alpha_i}{2\,l^2} \|g_i(f(k_1)) - g_i(f(k_2))\|_1).$$

Thus, $K_{SOT}(k_1, k_2)$ can be written as a product of Ornstein-Uhlenbeck kernels chained with functions $g_i \circ f$. Kernels chained with arbitrary mappings are kernels and products of kernels are kernels (see [16], Proposition 3.22). Thus, $K_{SOT}(k_1, k_2)$ is a proper (p.s.d.) kernel. ■

D Experimental Details

In this section, we give further details on parameter configuration and implementation details of the different methods.

Datasets: All datasets are publically available. Powerplant, Airfoil and Concrete are UCI regression datasets (`https://archive.ics.uci.edu/ml/datasets.php`). For LGBB and Airline, we describe how to access the datasets in the accompanying code at `https://github.com/boschresearch/bosot`.

SOT Kernel-Kernel (Our method): We use Algorithm 2 to optimize the acquisition function. Our base setting uses a population size of 100 and $n_{\text{offspring}} = 4$. The number of selected kernels is chosen such that the population stays constant. We choose 10 optimization steps for the bigger search spaces LGBB, Powerplant as well as Airfoil and 6 steps for the smaller search space Airline. We chose this number depending on the number of base kernels in the search space.

Hellinger Kernel-Kernel: For the Hellinger Kernel-Kernel [9], we use their principle of optimizing the acquisition function, where an active set of kernels is kept in memory over the iterations. In each iteration, 15 random walks are performed in the kernel grammar, where the walk length is drawn randomly from a geometric distribution with $p = \frac{1}{3}$. These kernels are added to the active set. Furthermore, the neighbors of the best kernel found so far are added to the active set (we limit this set to 50 neighbors per iteration, since for large expressions there may be several hundred neighbors, making it computationally almost infeasible to evaluate the kernel-kernel on all of them). The active set is limited to 600 kernels, where only the ones are kept with the highest acquisition function value. We initialize the active set with random walks in the grammar, with one random walk of length 5 from each base kernel (this is an alternative to their computationally very expensive version of using all kernels two edges apart from the base kernels). We stick to their base settings of using 100 hyperparameter samples and a subset of the design matrix $\mathbf{X}$ of size of 20 inside the Hellinger distance calculation. We cache all distance calculations $d(\mathcal{M}, \mathcal{M}')$ over the iterations, thus, recalculation of the Hellinger distance for kernels in the active set or the current dataset is very cheap. On the other hand, calculating the distance that involves a new kernel, such as a new neighbor of the currently best kernel, is very expensive. Given the distance matrix between kernels, optimization of the kernel-kernel hyperparameters is cheap again.

Greedy Search: The greedy method in [3] starts with the empty kernel, then evaluates all base kernels. It picks the best performing base kernel and determines its immediate neighbors via expanding the base kernels with all possible grammar operations. Then the neighbors are evaluated (we pick the order in which the neighbors are evaluated at random - for each seed a different order). Once all neighbors have been evaluated, the best kernel is determined and the process repeats. We give greedy search a head start of $n_{initial}$ log-evidence evaluations in Figure 1, meaning log-evidence values are shown once greedy search has evaluated as many kernels as the other methods have in their initial dataset.

TreeGEP: For the evolutionary algorithm in [4], we use their base settings in the paper, which is a population size of 200, a reproduction rate of 0.1, and a probability of mutation vs. cross-over of

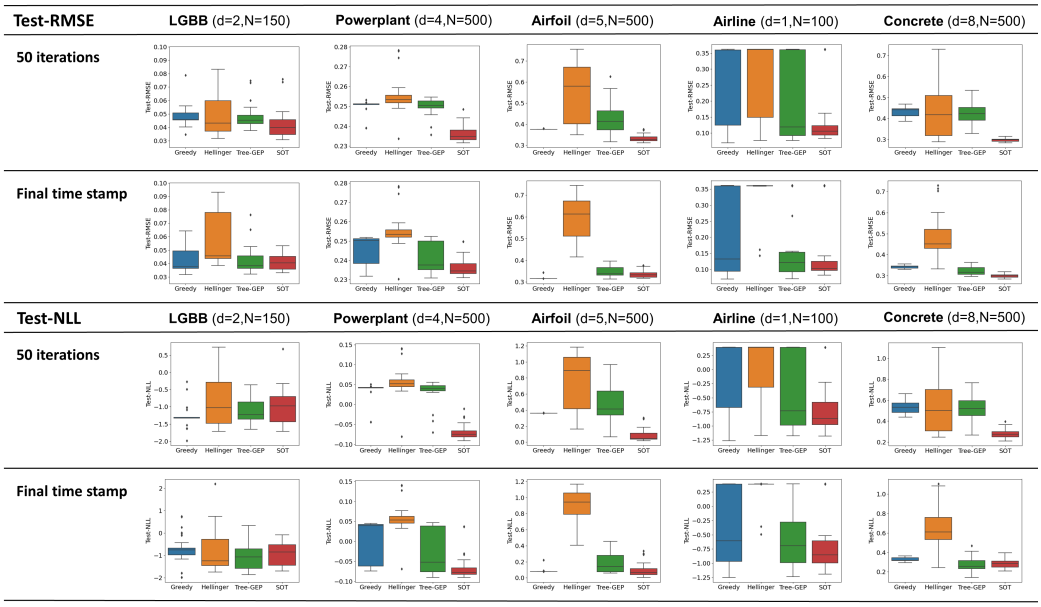


Figure 2: Box-plots of the test-RMSE and test-NLL of the selected hypotheses after 50 iterations and at the final time stamp.

50% each. They don't specify the size of the mutation subtrees and the tournament fraction in the tournament selection. Here, we make a reasonable choice and generate mutation subtrees of size four and use a tournament fraction of 0.1.

Test Performance - Box-Plots: In Figure 2, we show box-plots of the RMSE and NLL on the held out test-sets for the selected kernels. We see that on both metrics, our search method also finds a final model that often yields the best test-performance or is among the best models. However, we also note that test performance is mainly a property of the model selection criteria and how well models that maximize that criteria generalize to new data points.

Acquisition Function Optimization vs. Oracle Evaluation Ratio: In Table 3, the ratio $\frac{t_{\text{Acquisition}}}{t_{\text{Oracle}}}$ between the CPU-time needed for optimization of the acquisition function and the CPU-time for evaluating the oracle (calculating the log-evidence) is shown for the SOT kernel-kernel and the Hellinger kernel-kernel [9]. As described in the previous section, the approach in [9] uses a different kind of acquisition function optimization, that has fewer evaluations of the acquisition function than the evolutionary algorithm we use. Nevertheless, we are magnitudes of orders faster. This can be understood more clearly when looking at the raw kernel-kernel evaluation times in Figure 3. A faster acqusition to oracle time ratio in the end results in a faster search procedure, measured over CPU-Time (as we show in our main results in Figure 1), which is the metric we are interested in.

kNN for Meta Prediction: For the k-nearast-neighbour approach that was used in Table 1 we employ the same principle as in [9] who also consider kNN for comparision. The set $\mathbb{K}_{\text{complete}}$

Table 3: Acquisition to oracle time ratio.

Dataset	Hellinger	SOT (ours)
Airfoil($N = 500$)	18.77	0.25
Airline($N = 100$)	54.21	0.825
LGBB($N = 150$)	42.63	0.52
Powerplant($N = 500$)	16.51	0.278
Concrete($N = 500$)	16.81	0.213

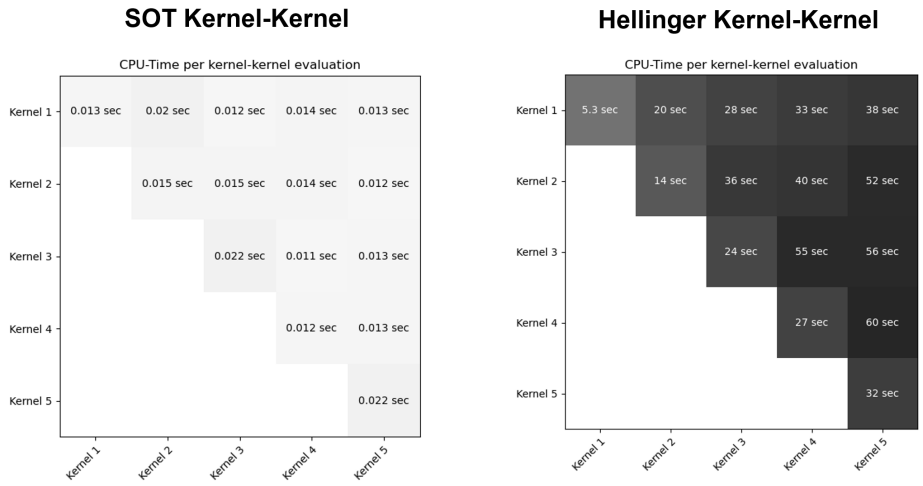


Figure 3: CPU-time for single kernel-kernel evaluations $K(k_1, k_2)$ for the SOT kernel-kernel and the Hellinger kernel-kernel.

Table 4: Summary of kernels used in Figure 3.

Kernel-Index	1	2	3	4	5
# Parameters	2	5	8	10	12
# Base Kernels	1	2	3	4	5

consists of kernel expressions which form a directed graph where two neighbour nodes are one grammar operation apart from each other. For each test expression $\tilde{k} \in \mathbb{K}_{\text{test}}$, we search in this graph the k expressions $\tilde{k}_1, \dots, \tilde{k}_k \in \mathbb{K}_{\text{train}}$ with shortest path in this directed graph. The prediction of the log-evidence value of $\tilde{k}$ is the average of the log-evidence values of $\tilde{k}_1, \dots, \tilde{k}_k$. The number k of neighbours is determined via cross validation.

Runs and Time Stamps: In Figure 1, all methods were repeated over 30 seeds, where each seed corresponds to a different initial dataset for the both BO methods and TreeGEP and a different ordering of neighbor evaluations in greedy search. All runs have different run-times. The reasons for this is that the oracle evaluation times differ depending on the kernel that is evaluated. The log-evidence can be calculated faster for smaller hypothesis. The final time stamps in Figure 1 are therefore determined by the shortest run of our method, as we only have log-evidence values for all runs/seeds up to that time point. In rare cases, a run can be interrupted, in case the Laplace approximation returns NaNs. This can happen due to numerical instabilities in the Cholesky decomposition and happened independently of the search method. We filtered out these runs.

E Further Experiments

Computational Time for Single Kernel-Kernel Evaluations: In Figure 3, we show CPU-time for single kernel-kernel evaluations $K(k_1, k_2)$ for the SOT kernel-kernel and the Hellinger kernel-kernel, each evaluated on five kernels generated from the kernel grammar (search space was the same as used for the LGBB dataset). We selected the five kernels with increasing numbers of base kernels and, thus, kernel parameters, as this could affect the computation time (see Table 4). Our kernel-kernel can be evaluated orders of magnitudes faster, which also explains the smaller acquisition function optimization times.

Kernel-Kernel Hyperparameters: In Figure 4, we show the values of the distance weights for the three OT metrics $W_{\tilde{d}}(\omega_{1,\text{base}}, \omega_{2,\text{base}})$ (on Airfoil summed over dimensions), $W_{\tilde{d}}(\omega_{1,\text{paths}}, \omega_{2,\text{paths}})$

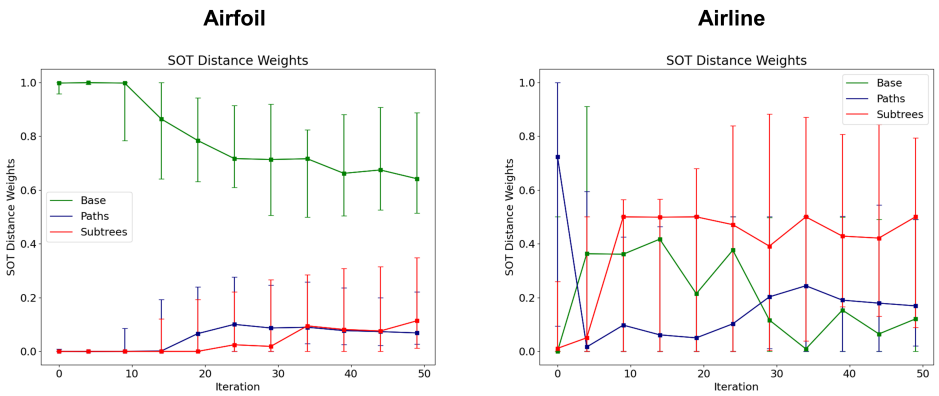


Figure 4: Distance weights of the SOT kernel-kernel over the BO iterations.

and $W_{\tilde{d}}(\omega_{1,\text{subtrees}}, \omega_{2,\text{subtrees}})$ over the BO iterations on Airline, and Airfoil. The weights were fitted via marginal likelihood maximization as described in A.5. First, we observe that all OT metrics are used. Secondly, we see that it is dependent on the dataset which OT metric is used primarily. For example, on Airline the Subtree features are the most important ones (according to the marginal likelihood maximization) whereas on Airfoil the Base features obtain the largest weights.

Interpretability of Final Hypothesis: The kernel grammar is used in a line of work called Automatic Statistician [3, 7]. In particular [7], utilize the kernel grammar to automatically generate natural language description of the data from the selected hypothesis (they also employ greedy search). Here, we show a configuration of our algorithm where the method of [7] can be applied to the final hypothesis of our model selection procedure and can give good descriptions of the dataset. As done in [7], we drop the rational quadratic kernel from the search space, as this captures small and long range correlations at the same time, which can also be modelled with separate squared exponential kernels (see [7]). Furthermore, we take fewer steps in the acquisition function optimizer such that fewer base kernels are maximally possible in the final hypothesis. This renders the final hypothesis smaller and easier to interpret, as fewer components needs to be described. We show two example hypotheses that were selected for the Airline dataset using the described search space and four steps in the acquisition function optimizer. We used the principles presented in [7] to simplify the expression and to generate the sentences.

Example 1:

$$\text{LIN} + \text{SE} + \text{LIN} \times \text{SE} + \text{PER} \times \text{LIN} \times \text{SE}$$

Desciption - Example 1: The data can be described as a sum of:

1. A linearly increasing function
2. A smooth function
3. A smooth function with increasing variation
4. An approx. periodic function with linearly increasing amplitude

Example 2:

$$\text{SE} + 2\,\text{LIN}^2 \times \text{PER} + 2\,\text{SE} \times \text{PER} \times \text{LIN}$$

Desciption - Example 2: The data can be described as a sum of:

1. A smooth function
2. Two approx. periodic functions with linearly increasing amplitude
3. Two periodic functions with quadratically increasing amplitude

Table 5: RMSE and predictive NLL values on the respective test-sets after 50 iterations of our proposed search method + the RMSE/NLL values of FKL [2] and a standard RBF kernel. SOT values are marked bold if they are not significantly different from the best value (FKL and RBF are point evaluations) according to a t-test ($\alpha = 0.05$).

Dataset	SOT (ours)	FKL	RBF	SOT (ours)	FKL	RBF
	RMSE			**NLL**		
Airline	**0.1335** (0.079)	0.3614	0.3598	**-0.7069** (0.442)	0.4712	0.3978
LGBB	**0.0422** (0.011)	0.1296	0.0740	**-0.9762** (0.517)	0.0200	**-1.0913**
Powerplant	**0.2362** (0.004)	0.2532	0.2507	**-0.0693** (0.019)	0.1755	0.0419
Airfoil	**0.3334** (0.017)	0.4356	0.4075	**0.0855** (0.081)	0.6469	0.2985
Concrete	**0.2980** (0.008)	0.3230	0.3617	**0.2787** (0.044)	4.2150	**0.2743**

Comparision with RBF Kernel: In Table 5 the test-set results (RMSE and NLL) of our proposed kernel search method is shown after 50 iterations compared to test values of a standard RBF kernel. We note that Table 5 shows different NLL values as Table 2 as it shows values at last iteration and not at last time stamp (see Appendix D for details). In terms of NLL, the RBF kernel shows competitive performance on the LGBB and Concrete dataset. However, considering the RMSE and NLL values on the other three datasets, it appears that kernel selection in general seems to be very important.

Comparision with Nonparametric Kernel Learning Methods: In Table 5 we also compare with the nonparametric kernel learning method, presented in [2], called Function Kernel Learning (FKL). FKL places a GP prior on the spectral density of the kernel - thus utilizing a nonparametric approach to kernel learning/selection. This results in a prior over spectral-mixture kernels, thus a prior over a fixed but highly flexible kernel structure. The results in Table 5 might be an indication that search over a discrete set of structural kernels might be beneficial, compared to placing a prior over a very flexible, but fixed kernel family.

Experiments on Simulated Data & Type-3 Maximum Likelihood Overfitting: In Figure 5, we show experimental results on simulated data. Here, we draw n datapoints from a GP prior with

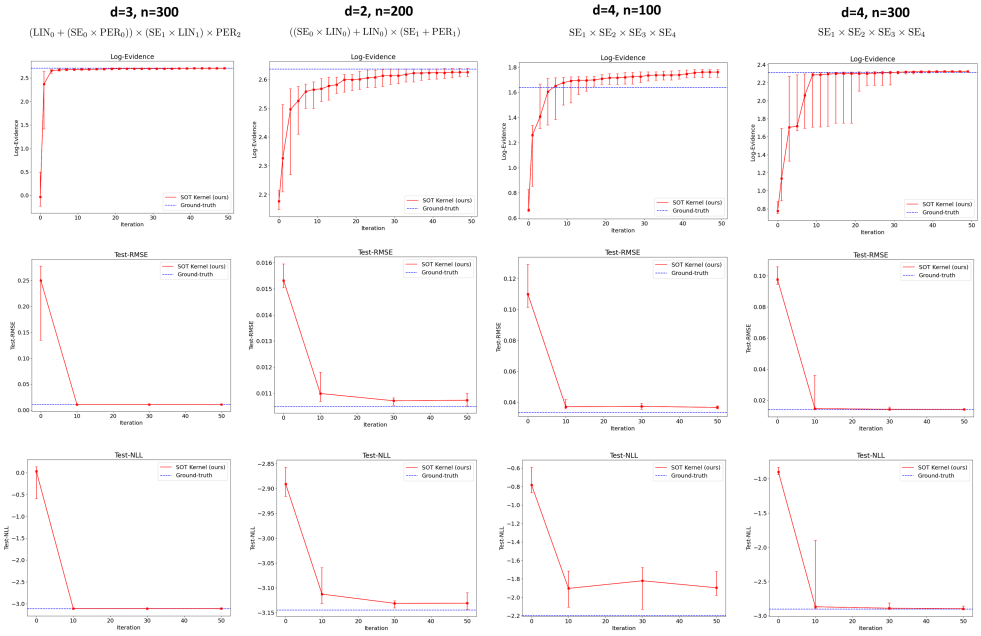


Figure 5: Learning-curves of our kernel selection method (over iterations) for simulated datasets. Upper plot shows log-evidence values, lower plot shows test-set RMSE and NLL values. Blue lines mark log-evidence/test values of the ground-truth kernel, from which the dataset was generated.

a given kernel structure and employ our kernel selection method on that dataset (likelihood noise was 0.01 for all datasets). First, we observe that we reach (almost) the same log-evidence values as the ground truth kernel within 50 iterations. For the RBF ground-truth kernel we simulated one big dataset and used $n = 100$ and $n = 300$ for model selection in the third and fourth plot. For the smaller dataset, we observe that we find kernels that have even higher log-evidence values than the ground-truth kernel. Considering the test-set NLL we also observe a small increase in the NLL from iteration 10 to 30, indicating a small overfitting. We think that in particular for smaller datasets maximizing the log-evidence over kernel structures can also lead to overfitting. However, this might not be surprising as the same was observed for type-2 maximum likelihood in GP's (see. [12]) whereas maximizing the log-evidence might be interpreted as type-3 maximum likelihood. Possible methods to avoid overfitting could be to use a different model selection criterion such as the Bayesian information criterion or cross-validation error, or to use a smaller search space.