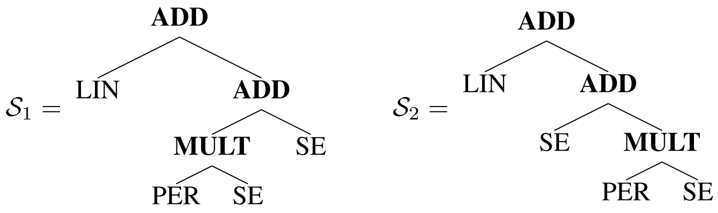
A Method Details

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.

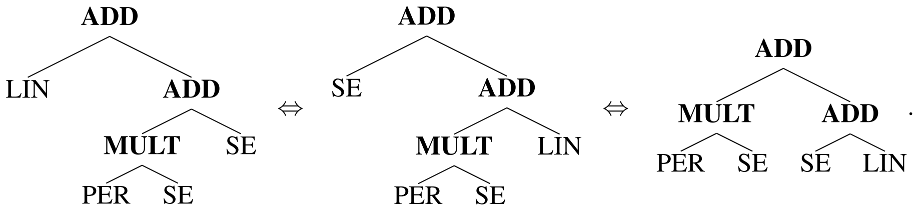
A.1 Symmetries in the Multisets

Our method can be used with general operators $T : \mathcal{K} \times \mathcal{K} \rightarrow \mathcal{K}$ 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:

$$\begin{aligned} &\mathbf{ADD} \longrightarrow \mathbf{ADD} \longrightarrow \mathbf{MULT} \longrightarrow \mathrm{PER}\,, \\ &\mathbf{ADD} \longrightarrow \mathbf{MULT} \longrightarrow \mathrm{PER}\,. \end{aligned}$$

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 $f : \mathbb{K} \rightarrow \mathcal{T}(\tilde{\mathbb{K}})$

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($\mathcal{D}$,$T$,$L$,$n_{\text{initial}}$):
    $\tilde{\mathcal{D}}_0 = \text{GetInitialDataset}(\mathcal{D}, n_{\text{initial}})$
    for $t = 0, \dots, T-1$ 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
    $t^* \leftarrow \arg\max_{t=0,\dots,T-1} g_t$
    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 $\tilde{\mathcal{D}} = \{(k_j, g(k_j|\mathcal{D}))|j = 1, \dots, R\}$ 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

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

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

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 $n_{\text{offspring}}$ children by generating $n_{\text{offspring}}$ 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.

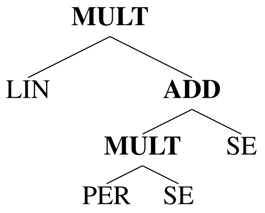
```
Function EvolutionaryAlg($a(\cdot|\tilde{\mathcal{D}}_t)$,$L$):
    $n_{\text{survive}} = \frac{n_{\text{population}}}{(n_{\text{offspring}}+1)}$
    $\mathcal{K}_0 = \text{GetInitialKernels}(n_{\text{population}})$
    for $l = 0, \dots, L-1$ do
        $\text{fitness}_l = a(\mathcal{K}_l|\tilde{\mathcal{D}}_t)$
        $\mathcal{K}_l^{\text{selected}} = \text{Select}(\mathcal{K}_l, \text{fitness}_l, n_{\text{survive}})$
        $\mathcal{K}_l^{\text{offspring}} = \emptyset$
        for $k$ in $\mathcal{K}_l^{\text{selected}}$ do
            $\mathcal{K}_l^{\text{offspring}} \leftarrow \mathcal{K}_l^{\text{offspring}} \cup \text{ApplyGrammarOps}(k, n_{\text{offspring}})$
        end
        $\mathcal{K}_{l+1} = \mathcal{K}_l^{\text{selected}} \cup \mathcal{K}_l^{\text{offspring}}$
    end
    return $\arg\max(a(\mathcal{K}_L|\tilde{\mathcal{D}}_t))$
```

Algorithm 2: Evolutionary algorithm for BO kernel search.

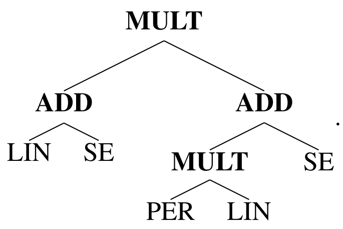
B Example Calculation of SOT Kernel-Kernel

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:

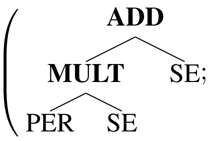
$\text{Base}(\mathcal{T}_1) = \{(\text{LIN}; 1), (\text{SE}; 2), (\text{PER}; 1)\},$

$\text{Base}(\mathcal{T}_2) = \{(\text{LIN}; 2), (\text{SE}; 2), (\text{PER}; 1)\},$

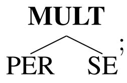
$$\begin{aligned}\text{Path}(\mathcal{T}_1) = \{&(\textbf{MULT} \longrightarrow \textbf{ADD} \longrightarrow \textbf{MULT} \longrightarrow \text{PER}; 1),\\ &(\textbf{MULT} \longrightarrow \textbf{ADD} \longrightarrow \textbf{MULT} \longrightarrow \text{SE}; 1),\\ &(\textbf{MULT} \longrightarrow \textbf{ADD} \longrightarrow \text{SE}; 1)\\ &(\textbf{MULT} \longrightarrow \text{LIN}; 1)\},\end{aligned}$$

$$\begin{aligned}\text{Path}(\mathcal{T}_2) = \{&(\textbf{MULT} \longrightarrow \textbf{ADD} \longrightarrow \textbf{MULT} \longrightarrow \text{PER}; 1),\\ &(\textbf{MULT} \longrightarrow \textbf{ADD} \longrightarrow \textbf{MULT} \longrightarrow \text{LIN}; 1),\\ &(\textbf{MULT} \longrightarrow \textbf{ADD} \longrightarrow \text{SE}; 2)\\ &(\textbf{MULT} \longrightarrow \textbf{ADD} \longrightarrow \text{LIN}; 1)\},\end{aligned}$$

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

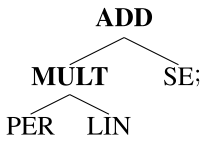


; 1), (

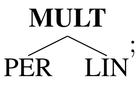


; 1), $(\text{LIN}; 1), (\text{SE}; 2), (\text{PER}; 1)\},$

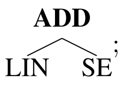
$\text{Subtree}(\mathcal{T}_2) = \{(\mathcal{T}_2; 1),$ (



; 1), (



; 1), (



; 1), $(\text{LIN}; 2),$

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

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

$$\omega_{1,\text{base}} = \frac{1}{4}\delta_{\text{LIN}} + \frac{1}{2}\delta_{\text{SE}} + \frac{1}{4}\delta_{\text{PER}},$$

$$\omega_{1,\text{base}} = \frac{2}{5}\delta_{\text{LIN}} + \frac{2}{5}\delta_{\text{SE}} + \frac{1}{5}\delta_{\text{PER}},$$

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

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

For the paths, the total variation distance results to:

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

For the subtrees, the total variation distance results to:

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

```
Function GetKernel($\mathcal{T}$):
    if Root of $\mathcal{T}$ is leaf then
        return base kernel $\mathcal{B}$ associated with leaf
    else
        return $T$(GetKernel($\mathcal{T}_L$), 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 $\{T_1, \dots, T_l\}$ and whose leafs are associated with the base kernels $\{\mathcal{B}_1, \dots, \mathcal{B}_r\}$ 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:

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

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

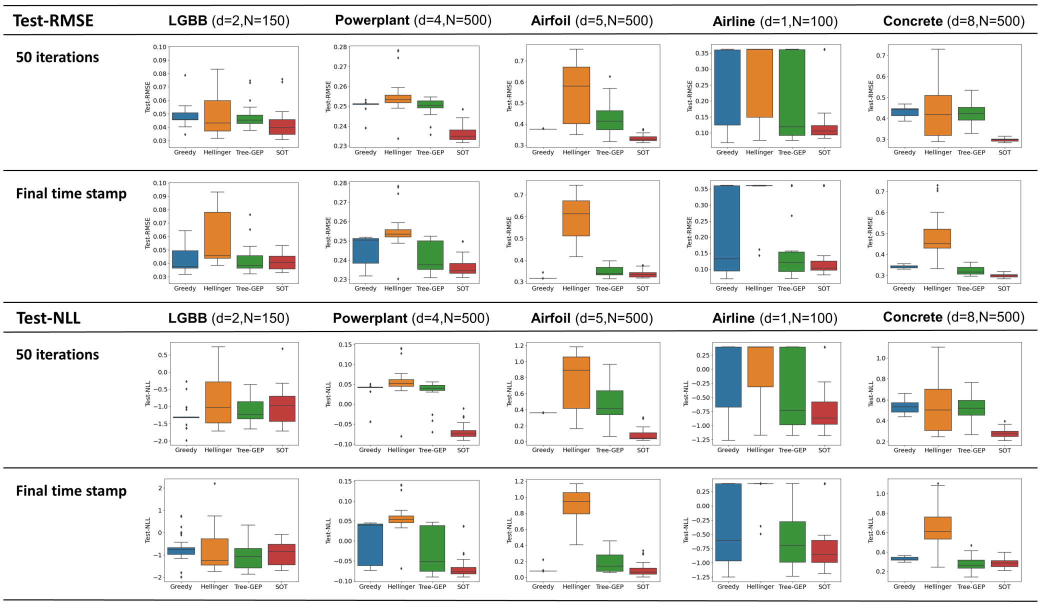


Figure 2: Box-plots of the test-RMSE and test-NLL of the selected hypotheses after 50 iterations and at the final time stamp.

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)
$\text{Airfoil}(N = 500)$	18.77	0.25
$\text{Airline}(N = 100)$	54.21	0.825
$\text{LGBB}(N = 150)$	42.63	0.52
$\text{Powerplant}(N = 500)$	16.51	0.278
$\text{Concrete}(N = 500)$	16.81	0.213

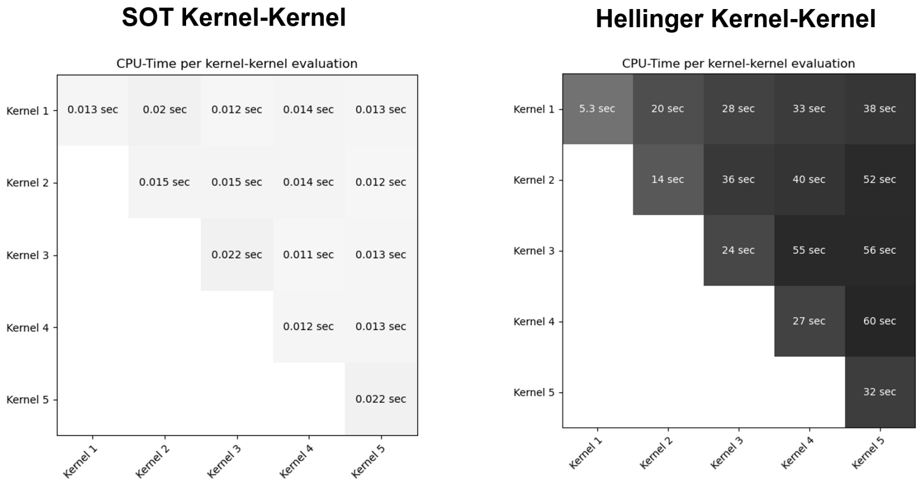


Figure 3: CPU-time for single kernel-kernel evaluations $K(k_1, k_2)$ for the SOT kernel-kernel and the Hellinger kernel-kernel.

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 $\tilde{k}_1, \ldots, \tilde{k}_k \in \mathbb{K}_{\text{train}}$ 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 $\tilde{k}_1, \ldots, \tilde{k}_k$. 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 $\mathrm{Airfoil}$ summed over dimensions), $W_{\tilde{d}}(\omega_{1,\text{paths}}, \omega_{2,\text{paths}})$

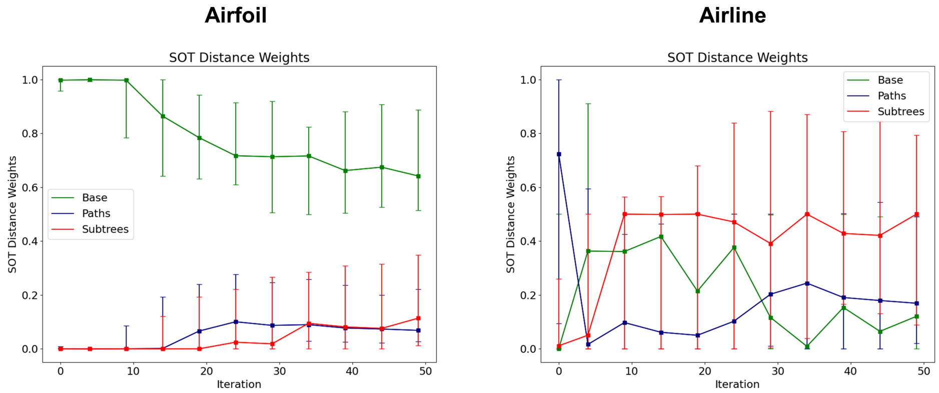


Figure 4: Distance weights of the SOT kernel-kernel over the BO iterations.

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:

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

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:

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

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

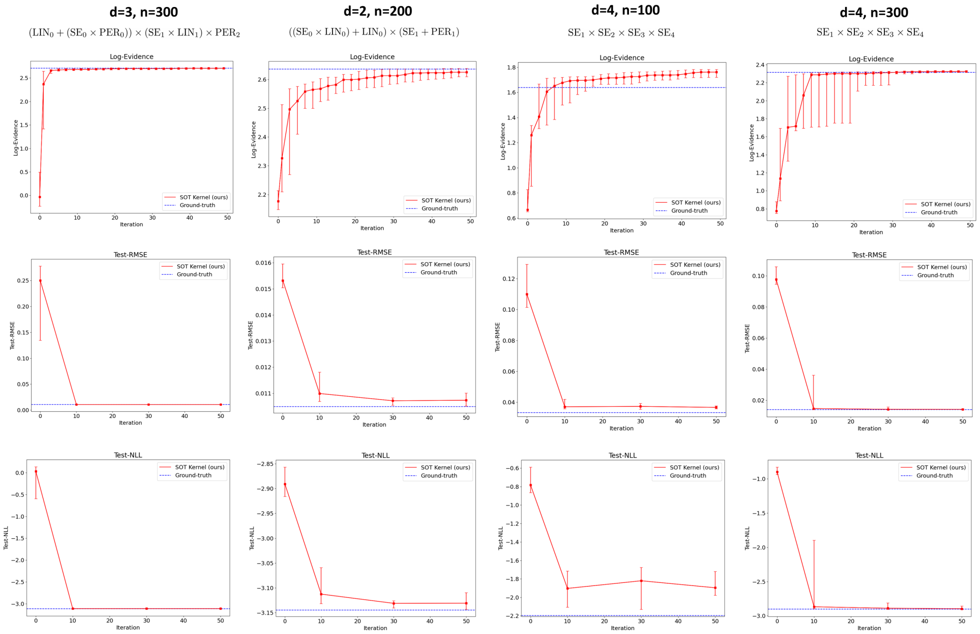


Figure 5: Learning-curves of our kernel selection method (over iterations) for simulated datasets. Upper plot shows log-evidence values, lower plot shows test-set RMSE and NLL values. Blue lines mark log-evidence/test values of the ground-truth kernel, from which the dataset was generated.

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.