Appendices

A Implementation Details

The most important part in our implementation is a linear representation of the tree structure using breadth-first search (BFS). When BFS encounters a node, the linearization routine adds a tag, which is the rank of this node in its siblings, in front of the variables associated with this node, and this tag will be used to construct the delta kernel in Algorithm 1. Figure 6 shows the liner representation corresponding to the tree structure in Figure 1.

0	$\boldsymbol{v}_r$	0	$\boldsymbol{v}_{p_1}$	1	$\boldsymbol{v}_{p_2}$

Figure 6: Linear representation of the tree structure corresponding to Figure 1

When linearizing an observation, we modify its tag using the following rules:

- if a node is not associated with a continuous parameter, its tag is changed to a unique value
- if a node doesn't in this observation's corresponding path, its tag is changed to a unique value
- otherwise, we set the values after this tag to be the sub-parameter restricted to this node

For example, the linear representation of an observation $(0.1, 0.2, 0.3, 0.4)$ falling into the left path is shown in Figure 7 and the linear representation of an observation $(0.5, 0.6, 0.7, 0.8, 0.9)$ falling into the right path is shown in Figure 8.

0	0.1	0.2	0	0.3	0.4	Uniq	$\boldsymbol{v}_{p_2}$

Figure 7: Linear representation of an observation falling into the lower path corresponding to Figure 1

0	0.5	0.6	Uniq	$\boldsymbol{v}_{p_1}$	1	0.7	0.8	0.9

Figure 8: Linear representation of an observation falling into the upper path corresponding to Figure 1

Based on this linear representation, it is now straightforward to compute the covariance function, which is constructed in Algorithm 1, between any two observations. We note the dimension of such a linear representation has order $\mathcal{O}(d)$, where d is the dimension of the original parameter space, thus there is little overhead compared with other covariance functions in existing GP libraries, such as RBF or Matern in GPyOpt.

B Time Complexity Analysis of Inference with Add-Tree

We analyse the time complexity of inference using our proposed Add-Tree covariance function by recursion. Without loss of generality, let the tree structure $\mathcal{T}$ be a binary tree. If the root node r of $\mathcal{T}$ is associated with a continuous parameter, which means this parameter is shared by all paths. In this worst case, the gram matrix in Equation (9) is dense and structureless, and the complexity will be $\mathcal{O}(n^3)$, where n is the number of observations. Otherwise, when r is not associated to any continuous parameter, let n_l and n_r be the number of samples falling into the left path and the right path respectively, we have $T(r) = T(rl) + T(rr)$, where rl is the left child of r, rr is the right child of r, $T(r), T(rl), T(rr)$ are the running time at nodes r, rl, rr respectively. Because the worst-case running time at nodes rl and rr is $\mathcal{O}(n_l^3)$ and $\mathcal{O}(n_r^3)$ respectively, we have $T(r) = \mathcal{O}(n_l^3 + n_r^3)$.

Table 2 summarizes the worst-case inference time complexity comparison of our Add-Tree covariance function and other related methods. In Table 2, n_i is number of observations falling into path l_i and $n = \sum_{1 \leq i \leq |P|} n_i^3$. In general, Add-Tree performs the worst among these three methods from the aspect of time complexity. However, due to the explicit sharing mechanism, our approach requires fewer black-box calls to the expensive objective function, which typically dominates the computational cost of the GP model.

C Time Complexity Analysis of Algorithm 2

W.l.o.g, let the tree structure $\mathcal{T}$ be a perfect binary tree, the depth of this tree be h, and suppose all nodes are associated with a d_u dimensional vector. Then $|P| = 2^{h-1}$ and $|V| = 2^h - 1$. The running time of searching in every leaves in a naïve way is $|P| + |P|\mathcal{O}(h^2 d_u^2) = 2^{h-1} + 2^{h-1}\mathcal{O}(h^2 d_u^2)$. The running time of Algorithm 2 is $|V|\mathcal{O}(d_u^2) + |P|h + |P| = 2^{h-1} + 2^{h-1}(h + 2\mathcal{O}(d_u^2)) - \mathcal{O}(d_u^2)$, here we keep the constants just for clarity. It is clear that Algorithm 2 has a substantial advantage over a naïve method when $h \geq 2$. We note the complexity of BFGS is $\mathcal{O}(n^2)$, where n is the dimension of the parameter.

Table 2: Inference Time Complexity Comparison

Method	**Share?**	**Complexity**	
Independent	no	$\mathcal{O}(\sum_{1\le i\le \lvert P\rvert} n_i^3)$	
Jenatton et al.	yes	$\mathcal{O}(\sum_{1\le i\le \lvert P\rvert} n_i^3 + \lvert V\rvert^3)$	
Add-Tree	yes	$\mathcal{O}(n_l^3 + n_r^3)$	if no sharing continuous parameter at r
		$\mathcal{O}(n^3)$	otherwise

D Combine With Other Acquisition Functions

Add-Tree covariance function itself can be combined with any other acquisition functions and enables efficient information sharing. To efficiently optimize the acquisition function using Algorithm 2, it is required the acquisition function has additive structure, otherwise the two-step approach in Jenatton et al. (2017) can be used.

E When Additive Assumption is Not Enough

For objective functions with known additive structure, our proposed Add-Tree covariance function usually performs the best. If there is an interaction effect between the variables along a single path in the tree structure, we can combine the method proposed in Duvenaud et al. (2011) by including higher order additive kernels for these variables. To illustrate the covariance function design in this case, we again take the tree-structured function in Figure 1 as an example. Since there is an interaction effect between $\boldsymbol{v}_r$ and $\boldsymbol{v}_{p1}$, the latent variables associated to $f_{p_1,\mathcal{T}}$ is decomposed as $\boldsymbol{f}_r^{(1)} + \boldsymbol{f}_1 + \boldsymbol{f}_{r1}$, where $\boldsymbol{f}_{r1}$ is the interaction term between $\boldsymbol{v}_r$ and $\boldsymbol{v}_{p1}$. Similarity, the latent variables associated to $f_{p_2,\mathcal{T}}$ is $\boldsymbol{f}_r^{(2)} + \boldsymbol{f}_2 + \boldsymbol{f}_{r2}$. Similar to Equation (6), we have:

$$\begin{bmatrix} \boldsymbol{f}_r^{(1)} + \boldsymbol{f}_1 + \boldsymbol{f}_{r1} \\ \boldsymbol{f}_r^{(2)} + \boldsymbol{f}_2 + \boldsymbol{f}_{r2} \end{bmatrix} = \begin{bmatrix} \boldsymbol{f}_r^{(1)} \\ \boldsymbol{f}_r^{(2)} \end{bmatrix} + \begin{bmatrix} \boldsymbol{f}_1 \\ \boldsymbol{f}_2 \end{bmatrix} + \begin{bmatrix} \boldsymbol{f}_{r1} \\ \boldsymbol{f}_{r2} \end{bmatrix}. \quad (12)$$

To model $\boldsymbol{f}_{r1}$ and $\boldsymbol{f}_{r2}$ separately, we can use product covariance functions $k_r k_1$ and $k_r k_2$ respectively. Without further assumptions, it is not clear how to model the covariance between $\boldsymbol{f}_{r1}$ and $\boldsymbol{f}_{r2}$. A visualization is shown in Equation (13). In this case, a safe choice is to set these covariance to be zero, because overestimating the covariance will confuse the GP, and the price paid for ignoring these covariance here is we lose some potential sample-efficiency.

$$\begin{bmatrix} \boldsymbol{f}_{r1} \\ \boldsymbol{f}_{r2} \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{array}{|c|c|} \hline K_{r1} & ? \\ \hline ? & K_{r2} \\ \hline \end{array}\right). \quad (13)$$

Combine Equations (9), (12) and (13), we obtain the joint distribution of a tree-structured function with interaction effects:

$$\begin{bmatrix} \boldsymbol{f}_r^{(1)} + \boldsymbol{f}_1 + \boldsymbol{f}_{r1} \\ \boldsymbol{f}_r^{(2)} + \boldsymbol{f}_2 + \boldsymbol{f}_{r2} \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K_{11} & K_r^{(12)} \\ K_r^{(21)} & K_{22} \end{bmatrix}\right), \quad (14)$$

where $K_{11} = K_r^{(11)} + K_1 + K_{r1}$ and $K_{22} = K_r^{(22)} + K_2 + K_{r2}$. To implement the Add-Tree covariance function with interaction effects, the linear representation presented in Appendix A remains unchanged. For Algorithm 1, we only need to construct an extra term by multiplying the corresponding delta covariance function with the interaction terms we are interested in, and append this extra term in the final covariance function.