A Appendix

B General experimental setup

All experimental results presented in Section 5 were evaluated on an HTCondor cluster (see [64]) of machines equipped with Intel Core i7-8700 3.20GHz and 16 GB RAM. Confidence intervals show the first and third quartile of 20 independent runs with random seeds $\in [101, 102, \ldots, 120]$. If not specifically indicated in Section C, all competing algorithms use default values for all hyperparameters. To allow for a fair comparison, we give the same set of initial data points to all tested methods for both constrained and unconstrained benchmark problems. A set of ten initial samples is used for the Section 5.4 CIFAR-NAS example due to its high dimensionality. Five initial points are used for the remaining benchmark problems.

C Algorithms

This section summarizes the different algorithms used for the Section 5 numerical studies. We give implementation details and hyperparameter settings to reproduce the presented results.

C.1 LEAF-GP and LEAF-GP-RND

The `LEAF-GP` uses LightGBM [37] for training gradient-boosted tree ensembles. All runs use the hyperparameter value *min_data_in_leaf* $= 1$ as the training dataset size needs to be at least twice the minimum number of data points a leaf is based on. The *min_data_in_leaf* default value of LightGBM is 20, which would cause run-time errors after initialization. We also set LightGBM hyperparameter *min_data_per_group* $= 1$. For the high-dimensional benchmark problem CIFAR-NAS, we set *max_depth* $= 5$ and *num_boost_rounds* $= 100$ for training the ensemble in LightGBM, referring to maximum interaction depth per decision tree and total number of trees in the ensemble, respectively. For all other benchmarks we use *max_depth* $= 3$ and *num_boost_rounds* $= 50$.

We implement the tree ensemble kernel as a non-stationary kernel in GPyTorch [24]. For deriving the posterior distribution, we use a Gaussian likelihood and standardize the target values of the data set. Section 3 introduces signal variance σ_0 and noise term σ_y as kernel hyperparameters which are fitted by maximizing the marginal log likelihood over 200 epochs using the Adam solver [38]. The hyperparameters are constrained by intervals according to $\sigma_0 \in [5\text{e}{-4}, 0.2]$ and $\sigma_y \in [0.05, 20.0]$.

For `LEAF-GP` the Section 4 acquisition optimization formulation is encoded using *gurobipy* [28] and solved using Gurobi 9. Runs are limited to 100 s if the solver finds a feasible solution and are continued otherwise.

Moreover, we set *heuristics* $= 0.2$ and activate the non-convex hyperparameter for benchmark problems with non-convex constraints. `LEAF-GP-RND` uses a sampling-based strategy that randomly evaluates the acquisition function at 2000 locations and selects the maximimum value.

C.2 GA

We use the standard Genetic Algorithm implementation of the pymoo [7] toolbox for evolutionary algorithms and change *population_size* $= 10$ given the small evaluation budget. Default values are used for all other hyperparameters.

C.3 SKOPT-GBRT and SKOPT-RF

We use the default implementation of Scikit-Optimize [31] with random forest and gradient-boosted trees base estimators for `SKOPT-RF` and `SKOPT-GBRT`, respectively. Default values are used for all hyperparameters.

C.4 SMAC

For `SMAC` [34] we utilize the most recent Python implementation SMAC3 [44] using random forest models. Moreover, we specify the hyperparameters *run_obj* $=$ '$quality$' and activate the deterministic

flag to allow for reproducibility. The Section 5.4 CIFAR-NAS and Section F VAE-NAS benchmark problems exhibit hierarchical search space relationships, i.e., hyperparameters of a specific layer are only relevant if the layer is active. We use `SMAC`'s `InCondition` function which allows for certain child features to be considered only if some parent features have certain values, e.g., the stride of layer n is only considered if number of layers is at least n. This allows `SMAC` to capture hierarchical relationships explicitly and to avoid evaluating multiple equivalent configurations. Default values are used for all other hyperparameters.

C.5 UCB-MATERN and EI-MATERN

The `UCB-MATERN` and `EI-MATERN` algorithms use the standard upper confidence bound and expected improvement implementations of BoTorch [2]. Before fitting a GP, we normalize data features and standardize data outputs. The upper confidence acquisition hyperparameter β is set to 1.96. We negate the target values and define *raw_samples* $= 200$ for the acquisition function maximization. For unconstrained cases, the acquisition optimizer uses 100 restarts. However, for constrained problems we limit the optimizer to five restarts due to extensive run-times caused by finding feasible solutions. Default values are used for all other hyperparameters.

D Benchmark problems

D.1 Unconstrained problems

Figure 3 shows the results for Hartmann (6D), Rastrigin (10D), Schwefel (10D) and Styblinski-Tang (10D) benchmark functions implemented according to Surjanovic and Bingham [63]. Table 1 summarizes number of dimensions and domain evaluated for unconstrained benchmark problems.

Table 1: Benchmark functions for local vs. global acquisition-function optimization tests. Table shows function name, number of dimensions (Dim.), and domain of input variables

Function	Dim.	Domain
Hartmann	6	$\mathbf{x} \in [0.0, 1.0]^6$
Rastrigin	10	$\mathbf{x} \in [-4.0, 5.0]^{10}$
Schwefel	10	$\mathbf{x} \in [-500.0, 500.0]^{10}$
Styblinski-Tang	10	$\mathbf{x} \in [-5.0, 5.0]^{10}$

D.2 Constrained problems

Fig. 8 presents results of benchmark problems with known constraints. Domain bounds without decimals indicate integer-valued variable types. Benchmark examples G1, G3, G4, G6, G7 and G10 are implemented according to Hedar [32]. The Alkylation benchmark [60] is adapted from an open-source implementation [62]. To compare methods that do not support specific input constraints, we penalize the black-box function output according to:

$$f_{\text{penalty}} = \lambda \left(\max(g(\mathbf{x}), 0)^2 + h(\mathbf{x})^2 \right), \tag{15}$$

where $g(\mathbf{x}) \leq 0$ and $h(\mathbf{x}) = 0$ are inequality and equality constraints, respectively. Maximizing the combined black-box output allows methods to find feasible solutions. Eq. (15) has the hyperparameter λ which weights the penalty, we test values $\lambda \in \{1, 10, 100\}$ and only plot the best run for all methods that rely on constraint penalization. In the tests conducted, `LEAF-GP` fully supports explicit input constraints. `UCB-MATERN` and `EI-MATERN` support linear inequality and equality constraints only. The evolutionary algorithm `GA` has built-in constraint consideration but does not guarantee feasible solutions. `LEAF-GP-RND`, `SKOPT-GBRT`, `SKOPT-RF` and `SMAC` rely on the Eq. (15) penalty function.

D.3 CIFAR-NAS

Table 3 gives more details on the Section 5.4 CIFAR-NAS (29D) benchmark problem. CIFAR-NAS (29D) has a total of 29 hyperparameters to tune, i.e., 1 continuous, 15 integer, 8 binary and

Table 2: Benchmark functions for constrained search space tests. Table shows function name and number of: dimensions (D), equality constraints (EC), and inequality constraints (IC). Values in brackets indicate the number of linear constraints which are natively supported by some of the algorithms. Domain bounds without decimals indicate integer-valued variables.

Function	D	IC	EC	Domain
G1	13	9 (9)	0 (0)	$\mathbf{x}_{\{0,\dots,8,12\}} \in [0.0, 1.0]$, $\mathbf{x}_{\{9,10,11\}} \in [0.0, 100.0]$
G3	5	0 (0)	1 (0)	$\mathbf{x} \in [0.0, 1.0]^5$
G4	5	6 (0)	0 (0)	$\mathbf{x}_0 \in [78.0, 102.0]$, $\mathbf{x}_1 \in [33.0, 45.0]$, $\mathbf{x}_{\{2,3,4\}} \in [27.0, 45.0]$
G6	2	2 (0)	0 (0)	$\mathbf{x}_0 \in [13.0, 100.0]$, $\mathbf{x}_1 \in [0.0, 100.0]$
G7	10	8 (3)	0 (0)	$\mathbf{x} \in [-10.0, 10.0]^{10}$
G10	8	6 (3)	0 (0)	$\mathbf{x}_0 \in [100.0, 10.0\text{K}]$, $\mathbf{x}_{\{1,2\}} \in [1.0\text{K}, 10.0\text{K}]$, $\mathbf{x}_{\{3,\dots 7\}} \in [10.0, 1.0\text{K}]$
Alkylation	7	14 (0)	0 (0)	$\mathbf{x}_0 \in [0.0, 2.0\text{K}]$, $\mathbf{x}_1 \in [0.0, 16.0\text{K}]$, $\mathbf{x}_2 \in [0.0, 120.0]$, $\mathbf{x}_3 \in [0.0, 5.0\text{K}]$, $\mathbf{x}_4 \in [90.0, 95.0]$, $\mathbf{x}_5 \in [0.01, 4.0]$, $\mathbf{x}_6 \in [145.0, 162.0]$
Pressure Vessel	4	3 (2)	0 (0)	$\mathbf{x}_{\{0,1\}} \in [1, 99]$, $\mathbf{x}_{\{2,3\}} \in [10.0, 200.]$

5 categorical variables. The goal is to select hyperparameter values for a CNN trained in PyTorch [55] on the CIFAR-10 dataset [40] that maximize test accuracy. The training and test scripts were adapted from Trencseni [70]. Due to limited computing resources, we train the CNN on half of the training data for 10 epochs using the Adam solver [38]. We score networks using the full test data set. Only certain combinations of stride, padding and filter size for various layers result in feasible neural architectures, e.g., the filter size of one layer may be too large given the output of the previous layer. In such cases the CNN training fails, and the black-box returns the largest black-box value found so far, helping algorithms learn to avoid infeasible neural architectures. To simplify the training, layer inputs are parameterized based on outputs of the previous layer for convolutional and fully-connected layers, as well as the intermediate connecting layer. The benchmark introduces categorical variables for activation function selection. Methods that do not support categorical features use one-hot encoding.

`LEAF-GP` has access to constraints capturing feasible neural architectures mainly concerned with the convolutional layers. Algorithms can choose to activate at most three convolutional and two fully-connected layers. To capture constraints for feasible CNNs, we introduce $w_{\text{out},i}$ as the output of convolutional layer i and $W_{\text{out},i}$ as the layer's modified output in case max-pooling is applied:

$$w_{\text{out},i} \in \mathbb{N}_0, \forall i \in \{1, 2, 3\} \tag{16a}$$
$$W_{\text{out},i} \in \mathbb{N}_0, \forall i \in \{1, 2, 3\} \tag{16b}$$

Convolutional layers use PyTorch's `Conv2D` with inputs derived by the optimization algorithms. PyTorch's `MaxPool2d` implements the max-pooling with the commonly-used $(2, 2)$ kernel size.

$$W_{\text{in},1} = 32 \tag{17a}$$
$$w_{\text{out},1} = \frac{W_{\text{in},1} - F_1 + 2P_1}{S_1} + 1 \tag{17b}$$
$$W_{\text{out},1} = b_1^{\text{conv}} \lfloor w_{\text{out},1}(1 - 0.5 b_1^{\text{pool}}) \rfloor + (1 - b_1^{\text{conv}}) W_{\text{in},1} \tag{17c}$$
$$w_{\text{out},2} = \frac{W_{\text{out},1} - F_2 + 2P_2}{S_2} + 1 \tag{17d}$$
$$W_{\text{out},2} = b_2^{\text{conv}} \lfloor w_{\text{out},2}(1 - 0.5 b_2^{\text{pool}}) \rfloor + (1 - b_1^{\text{conv}}) W_{\text{out},1} \tag{17e}$$

$$w_{\text{out},3} = \frac{W_{\text{out},2} - F_3 + 2P_3}{S_3} + 1 \tag{17f}$$

$$W_{\text{out},3} = b_3^{\text{conv}} \lfloor w_{\text{out},3}(1 - 0.5 b_3^{\text{pool}}) \rfloor + (1 - b_3^{\text{conv}}) W_{\text{out},2} \tag{17g}$$

$$W_{\text{out},3} \geq 1 \tag{17h}$$

$$1 \leq b_1^{\text{conv}} + b_2^{\text{conv}} + b_3^{\text{conv}} + b_1^{\text{fc}} + b_2^{\text{fc}} \tag{17i}$$

The 32×32 image size of CIFAR-10 data defines the input to the full CNN ($W_{\text{in},1}$) in Eq. (17a). Eq. (17b) combines filter size F_1, padding P_1 and stride S_1 of the first convolutional layer to compute its output size $w_{\text{out},1}$. Variable $W_{\text{out},1}$ captures the final output of the convolutional layer by considering if the layer is activated, i.e., $b_1^{\text{conv}} = 1$, and if max-pooling is applied, i.e., $b_1^{\text{pool}} = 1$. Constraints (17c)–(17g) denote the same restrictions for subsequent layers, each using the output size of the previous layer as its input size. Eq. (17h) ensures that the output of the last convolutional layer $W_{\text{out},3}$ is at least one. We also enforce that at least one layer be active, which is captured by Eq. (17i).

To break symmetries in the benchmark problems, we introduce Constraints (18a)–(18g):

$$b_3^{\text{conv}} \leq b_2^{\text{conv}} \leq b_1^{\text{conv}} \tag{18a}$$

$$\neg b_i^{\text{conv}} \rightarrow \neg b_i^{\text{pool}}, \forall i \in \{1,2,3\} \tag{18b}$$

$$\neg b_i^{\text{conv}} \rightarrow C_i^{\text{conv}} \leq 4, \forall i \in \{1,2,3\} \tag{18c}$$

$$\neg b_i^{\text{conv}} \rightarrow F_i \leq 2, \forall i \in \{1,2,3\} \tag{18d}$$

$$\neg b_i^{\text{conv}} \rightarrow S_i \leq 1, \forall i \in \{1,2,3\} \tag{18e}$$

$$\neg b_i^{\text{conv}} \rightarrow P_i \leq 0, \forall i \in \{1,2,3\} \tag{18f}$$

$$\neg b_i^{\text{conv}} \rightarrow Act_i^{\text{conv}} = \text{ReLU}, \forall i \in \{1,2,3\} \tag{18g}$$

Eq. (18a) activates layers in a particular order, and Eq. (18b) deactivates max-pooling when the associated convolutional layer is inactive. Constraints (18c)–(18g) set layer-specific hyperparameters to pre-defined default values when the associated layer is inactive. We select these defaults as the lower bound for non-categorical variables and the first category for categorical variables.

Constraints (19) express the same restrictions for fully-connected layers:

$$b_2^{\text{fc}} \leq b_1^{\text{fc}} \tag{19a}$$

$$\neg b_i^{\text{fc}} \rightarrow N_i^{\text{fc}} \leq 4, \forall i \in \{1,2\} \tag{19b}$$

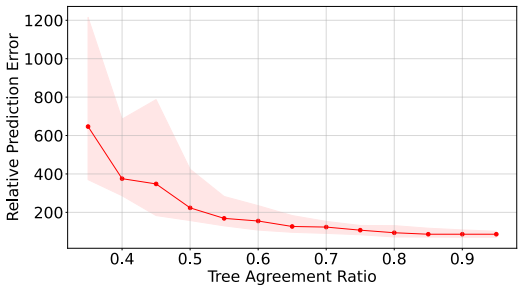
$$\neg b_i^{\text{fc}} \rightarrow Act_i^{\text{fc}} = \text{ReLU}, \forall i \in \{1,2\} \tag{19c}$$

Table 3: Table shows all hyperparameter names, types and domains of the CIFAR-NAS benchmark. The transformation column refers to post-processing computations before passing the hyperparameter value to the neural network training.

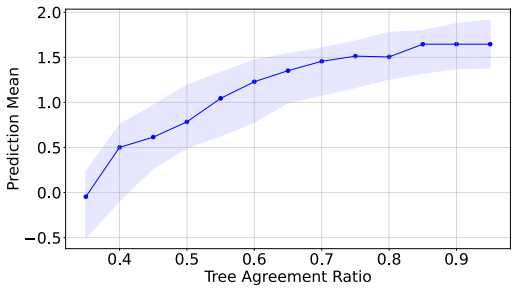
#	Name	Type	Domain	Transformation
0	Batch size	integer	$[2, 4]$	$N_{\text{batch}} = 2^{x_0}$
1	Learning rate	conti.	$[-5.0, -1.0]$	$\alpha = 10^{x_1}$
	Convolutional layer 1			
2	Layer is active	binary	$\{0, 1\}$	$b_1^{\text{conv}} = x_2$
3	Number of channels	integer	$[2, 4]$	$C_1^{\text{conv}} = 2^{x_3}$
4	Max pooling is active	binary	$\{0, 1\}$	$b_1^{\text{pool}} = x_4$
5	Filter size	integer	$[2, 5]$	$F_1 = x_5$
6	Stride	integer	$[1, 3]$	$S_1 = x_6$
7	Padding	integer	$[0, 3]$	$P_1 = x_7$
8	Activation function	categ.	$\{\text{ReLU}, \text{PReLU}, \text{Leaky ReLU}\}$	$Act_1^{\text{conv}} = x_8$
	Convolutional layer 2			
9	Layer is active	binary	$\{0, 1\}$	$b_2^{\text{conv}} = x_9$
10	Number of channels	integer	$[2, 4]$	$C_2^{\text{conv}} = 2^{x_{10}}$
11	Max pooling is active	binary	$\{0, 1\}$	$b_2^{\text{pool}} = x_{11}$
12	Filter size	integer	$[2, 5]$	$F_2 = x_{12}$
13	Stride	integer	$[1, 3]$	$S_2 = x_{13}$
14	Padding	integer	$[0, 3]$	$P_2 = x_{14}$
15	Activation function	categ.	$\{\text{ReLU}, \text{PReLU}, \text{Leaky ReLU}\}$	$Act_2^{\text{conv}} = x_{15}$
	Convolutional layer 3			
16	Layer is active	binary	$\{0, 1\}$	$b_3^{\text{conv}} = x_{16}$
17	Number of channels	integer	$[2, 4]$	$C_3^{\text{conv}} = 2^{x_{17}}$
18	Max pooling is active	binary	$\{0, 1\}$	$b_3^{\text{pool}} = x_{18}$
19	Filter size	integer	$[2, 5]$	$F_3 = x_{19}$
20	Stride	integer	$[1, 3]$	$S_3 = x_{20}$
21	Padding	integer	$[0, 3]$	$P_3 = x_{21}$
22	Activation function	categ.	$\{\text{ReLU}, \text{PReLU}, \text{Leaky ReLU}\}$	$Act_3^{\text{conv}} = x_{22}$
	Fully-connected layer 1			
23	Layer is active	binary	$\{0, 1\}$	$b_1^{\text{fc}} = x_{23}$
24	Number of nodes	integer	$[2, 7]$	$N_1^{\text{fc}} = 2^{x_{24}}$
25	Activation function	categ.	$\{\text{ReLU}, \text{PReLU}, \text{Leaky ReLU}\}$	$Act_1^{\text{fc}} = x_{25}$
	Fully-connected layer 2			
26	Layer is active	binary	$\{0, 1\}$	$b_2^{\text{fc}} = x_{26}$
27	Number of nodes	integer	$[2, 7]$	$N_2^{\text{fc}} = 2^{x_{27}}$
28	Activation function	categ.	$\{\text{ReLU}, \text{PReLU}, \text{Leaky ReLU}\}$	$Act_2^{\text{fc}} = x_{28}$

E Additional Results

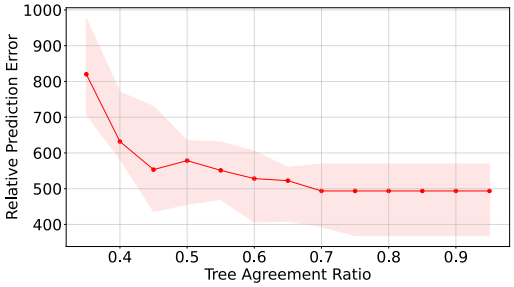
This section presents additional results supporting the numerical evaluation in Section 5.



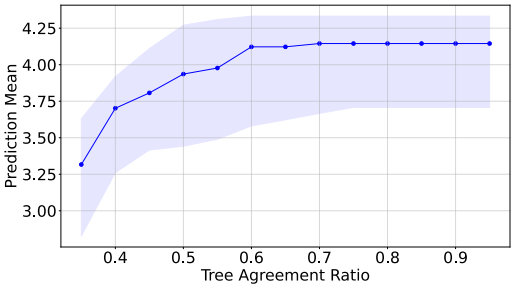
(a) Relative Model Error, Rastrigin (10D)



(b) Prediction Mean, Rastrigin (10D)

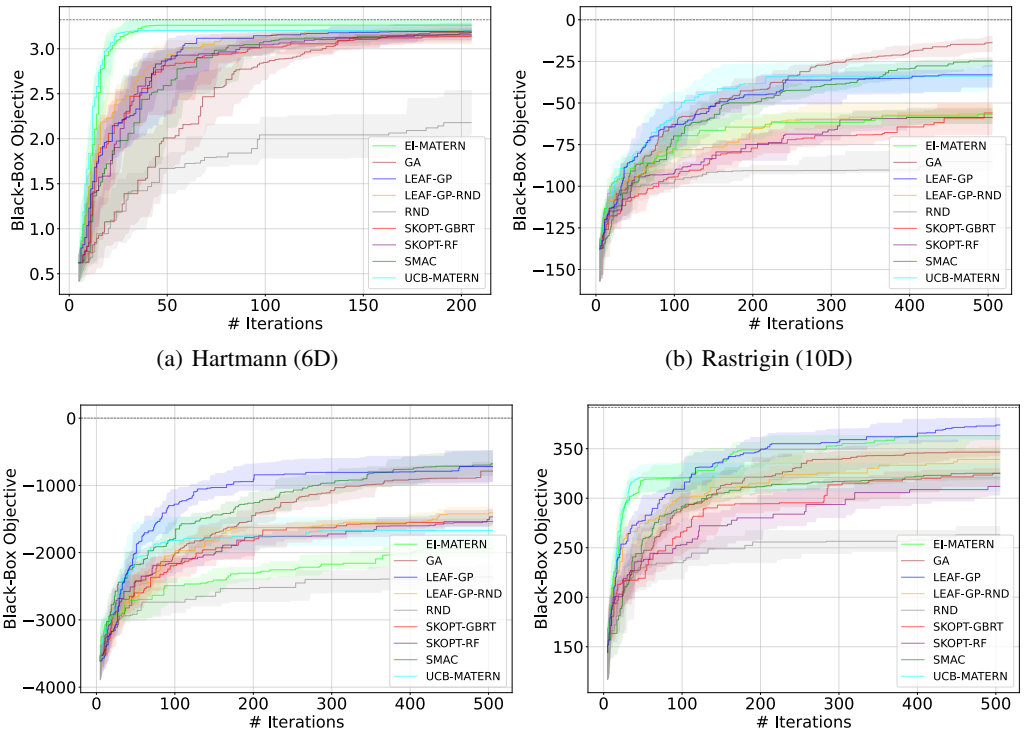


(c) Relative Model Error, Schwefel (10D)



(d) Prediction Mean, Schwefel (10D)

Figure 6: The relative prediction error (Eq. 13) and model prediction mean over the maximum tree agreement ratio R for benchmark problem Schwefel (10D). Changing R is equivalent to changing the maximum kernel covariance. Plot shows the median line and confidence intervals (first and third quartile) from 20 random seeds. Section 5.1 provides more details.



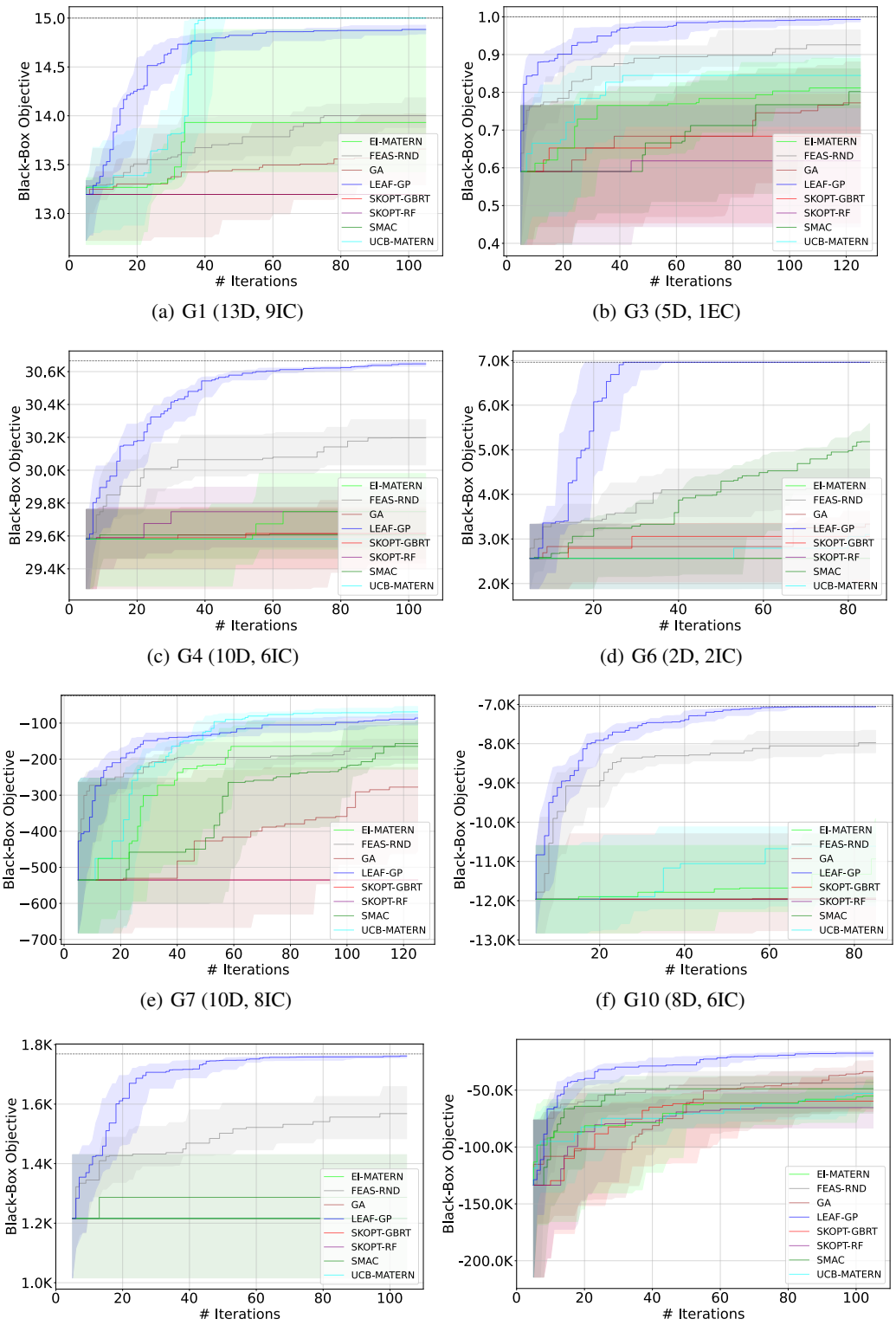
(a) Hartmann (6D)

(b) Rastrigin (10D)

(c) Schwefel (10D)

(d) Styblinski-Tang (10D)

Figure 7: Black-box optimization progress of `LEAF-GP` vs. baseline. Plot shows the median line and confidence intervals (first and third quartile) from 20 random seeds. Section 5.2 provides more details



(a) G1 (13D, 9IC)

(b) G3 (5D, 1EC)

(c) G4 (10D, 6IC)

(d) G6 (2D, 2IC)

(e) G7 (10D, 8IC)

(f) G10 (8D, 6IC)

(g) Alkylation (7D, 14IC)

(h) Pressure Vessel (4D, 3IC)

Figure 8: Feasible black-box optimization progress of `LEAF-GP` vs. baseline. Plot shows the median line and confidence intervals (first and third quartile) from 20 random seeds. Confidence intervals are neglected for methods that cannot improve the initial training data. Figure subtitles give the function name and number of: dimensions (D), equality constraints (EC), and inequality constraints (IC). Section 5.3 provides more details.

F VAE-NAS

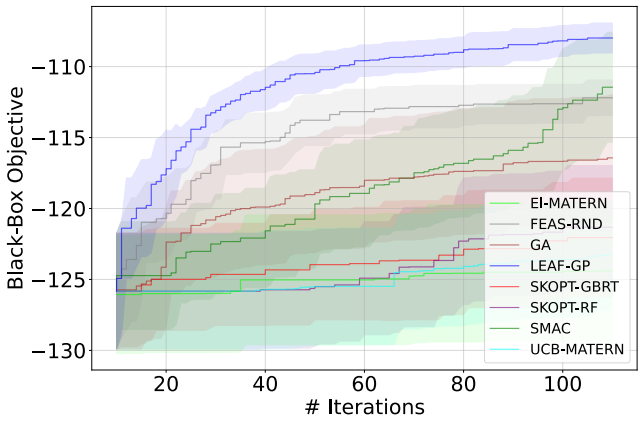


Figure 9: Feasible black-box optimization comparing `LEAF-GP` vs. baseline. Plot shows median line and confidence intervals (first and third quartiles) from 20 random seeds. Figure subtitles give the number of dimensions (D) and inequality constraints (IC). Section 5.4 provides details.

Table 4 gives more details on the Section 5.4 Variational Autoencoder Neural Architecture Search (VAE-NAS) benchmark problem, which was adapted from Daxberger et al. [16]. As their exact implementation is not publicly available, we created a benchmark problem based on the paper description and training scripts by Rath [58]. VAE-NAS (32D) has a total of 32 hyperparameters to tune, i.e., 1 continuous, 20 integer, and 11 categorical variables. The goal is to select hyperparameter values for a variational autoencoder (VAE) [39] trained in PyTorch [55] on the MNIST dataset [17] that minimize test loss, i.e., the average loss of when encoding and decoding images from the test set. We train each tested VAE for 32 epochs using the Adam solver [38] and a fixed batch size of 128. Only certain combinations of stride, padding, and filter size for various layers result in feasible neural architectures. The convolutional layer output size W^e_{out} is computed as:

$$W^e_{\text{out}} = \frac{W^e_{\text{in}} - F^{\text{e}} + 2P^{\text{e}}}{S^{\text{e}}} + 1 \in \mathbb{N}, \tag{20}$$

Decoder constraints are more complicated given that the size of the VAE output must match the original MNIST image size. The output of deconvolutional layers is computed according to:

$$W^d_{\text{out}} = S^{\text{d}}(W^d_{\text{in}} - 1) + F^{\text{d}} - 2P^{\text{d}} + O^{\text{d}} \tag{21}$$

Matching the output and input sizes of the VAE is non-trivial, as hyperparameters of the convolutional and deconvolutional layers are themselves set by the optimization algorithms. To simplify the training, layer inputs are parameterized based on outputs of the previous layer for convolutional, deconvolutional, and fully-connected layers. Specifically, fully-connected layers `FC2` and `FC3` have $2 \times N^{\text{lat}}$ nodes. According to Daxberger et al. [16], we parameterize the size of the last fully-connected layer—which can be `FC3`, `FC4`, or the latent space layer depending on which layers are active—as $C^{\text{d}}_1 \times 7 \times 7$. This allows methods not supporting explicit input constraints to still easily find a feasible neural architecture by deactivating all deconvolutional layers and computing an output $16 \times 7 \times 7 = 784$ with $C^{\text{d}}_1 = 16$ (the original MNIST image size is $1 \times 28 \times 28 = 784$). This rule supersedes other layer size definitions for fully-connected layers. Moreover, we extend the benchmark by introducing the nonlinear activation function of each layer as a categorical optimization hyperparameter. Methods that do not support categorical features use one-hot encoding. The activation function of the last layer is fixed as the sigmoid function, superseding other activation function hyperparameters. The VAE is trained using the sum of binary cross-entropy loss (reconstruction error) and KL divergence. We use the same loss function to evaluate the VAE's performance on the test dataset, giving the black-box objective in Fig. 9. To allow for a fair comparison, the same 10 randomly sampled feasible architectures initialize all methods.

`LEAF-GP` has access to constraints describing feasible neural architectures. We begin by defining auxiliary variables similar to the benchmark in Section D.3:

$$W^{\text{e}}_{\text{in},i} \in \mathbb{N}_0, \qquad \forall i \in [1, 2]\,, \tag{22a}$$

$$w^{\text{e}}_{\text{out},i} \in \mathbb{N}_0, \qquad \forall i \in [1,2]\,, \tag{22b}$$
$$W^{\text{e}}_{\text{out},i} \in \mathbb{N}_0, \qquad \forall i \in [1,2]\,, \tag{22c}$$
$$b^{\text{e}}_{\text{covn},i} \in \{0,1\}, \qquad \forall i \in [1,2]\,, \tag{22d}$$
$$N^{\text{e}}_{\text{conv}} = b^{\text{e}}_{\text{conv},1} + b^{\text{e}}_{\text{conv},2}, \tag{22e}$$
$$b^{\text{e}}_{\text{conv},1} \geq b^{\text{e}}_{\text{conv},2} \tag{22f}$$

where $W^{\text{e}}_{\text{in},i}$ and $W^{\text{e}}_{\text{out},i}$ denote, respectively, the input and output sizes of convolutional layer i in the encoder. We also define an auxiliary variable $w^{\text{e}}_{\text{out},i}$ to track the output size for inactive layers, as well as binary variables $b^{\text{e}}_{\text{conv},i}$ corresponding to the active/inactive state of each layer. Eq. (22e) and Eq. (22f) link binary variables $b^{\text{e}}_{\text{conv},i}$ to the number of active convolutional layers in the encoder. Using these auxiliary variables, the following relations can be expressed:

$$W^{\text{e}}_{\text{in},1} = 28, \tag{23a}$$
$$W^{\text{e}}_{\text{in},2} = W^{\text{e}}_{\text{out},1}, \tag{23b}$$
$$w^{\text{e}}_{\text{out},i} = \frac{W^{\text{e}}_{\text{in},i} - F^{\text{e}}_i + 2P^{\text{e}}_i}{S^{\text{e}}_i} + 1, \qquad \forall i \in [1,2]\,, \tag{23c}$$
$$W^{\text{e}}_{\text{out},i} = b^{\text{e}}_{\text{conv},i} w^{\text{e}}_{\text{out},i} + (1 - b^{\text{e}}_{\text{conv},i}) W^{\text{e}}_{\text{in},i}, \qquad \forall i \in [1,2]\,, \tag{23d}$$
$$W^{\text{e}}_{\text{out},2} \geq 1 \tag{23e}$$

Eq. (23a) and Eq. (23b) define the input sizes as the MNIST image input size $W^{\text{e}}_{\text{in},1} = 28$ for the first layer and the output size of the previous convolution for ensuing layers. Eq. (23c) defines the layer i output $w^{\text{e}}_{\text{out},i}$ given the filter size F^{e}_i, padding P^{e}_i, and stride S^{e}_i. Eq. (23d) ensures that the actual convolutional layer output $W^{\text{e}}_{\text{out},i}$ only takes the value of $w^{\text{e}}_{\text{out},i}$ if the layer is active. Finally, Eq. (23e) enforces the output size of the encoder to be at least one.

We add similar auxiliary variables and constraints for the deconvolutional layers:

$$W^{\text{d}}_{\text{in},i} \in \mathbb{N}_0, \qquad \forall i \in [1,2]\,, \tag{24a}$$
$$w^{\text{d}}_{\text{out},i} \in \mathbb{N}_0, \qquad \forall i \in [1,2]\,, \tag{24b}$$
$$W^{\text{d}}_{\text{out},i} \in \mathbb{N}_0, \qquad \forall i \in [1,2]\,, \tag{24c}$$
$$b^{\text{d}}_{\text{dec},i} \in \{0,1\}, \forall i \in [1,2]\,, \tag{24d}$$
$$N^{\text{d}}_{\text{dec}} = b^{\text{e}}_{\text{dec},1} + b^{\text{e}}_{\text{dec},2}, \tag{24e}$$
$$b^{\text{d}}_{\text{dec},1} \geq b^{\text{e}}_{\text{dec},2}, \tag{24f}$$
$$b^{\text{d}}_{\text{dec},1} \rightarrow W^{\text{d}}_{\text{out},2} = 28, \tag{24g}$$
$$\neg b^{\text{d}}_{\text{dec},1} \rightarrow C^{\text{d}}_1 = 16 \tag{24h}$$

The Eq. (24g) indicator constraint restricts the decoder output to be the original image size $W^{\text{d}}_{\text{out},2} = 28$ if deconvolutional layers are active. Another indicator constraint Eq. (24h) handles the aforementioned case where no deconvolutional layer is active and $C^{\text{d}}_1 = 16$ ensures that the decoder output size can be resized to original MNIST image size. We emphasize that this rule is introduced to simplify the feasible architecture search for methods that do not support explicit input constraints. Note that `LEAF-GP` could add additional constraints to ensure the architecture's output size can always be resized to the original image size of 28×28.

$$W^{\text{d}}_{\text{in},1} = 7, \tag{25a}$$
$$W^{\text{d}}_{\text{in},2} = W^{\text{d}}_{\text{out},1}, \tag{25b}$$
$$w^{\text{d}}_{\text{out},i} = S^{\text{d}}_i (W^{\text{d}}_{\text{in},i} - 1) + F^{\text{d}}_i - 2P^{\text{d}}_i + O^{\text{d}}_i, \qquad \forall i \in [1,2]\,, \tag{25c}$$
$$S^{\text{d}}_i \geq O^{\text{d}}_i + 1, \qquad \forall i \in [1,2]\,, \tag{25d}$$
$$W^{\text{d}}_{\text{out},i} = b^{\text{d}}_{\text{conv},i} w^{\text{d}}_{\text{out},i} + (1 - b^{\text{d}}_{\text{conv},i}) W^{\text{d}}_{\text{in},i}, \qquad \forall i \in [1,2] \tag{25e}$$

Similar to the encoder, Eq. (25) defines constraints for feasible decoder layers. For deconvolutional layers, we also tune output padding O^{d}_i. According to the PyTorch [24] documentation, output padding must be smaller than either stride or dilation. Given that we do not optimize dilation in

deconvolutional layers, Eq. (25d) enforces output padding to be smaller than stride. We introduce similar constraints for fully-connected layers in both the encoder and decoder:

$$b_{\text{fc},i} \in \{0,1\}, \qquad \forall i \in [1,4]\,, \tag{26a}$$
$$N_{\text{fc}} = b_{\text{fc},1} + b_{\text{fc},2}, \tag{26b}$$
$$N^{\text{d}}_{\text{fc}} = b_{\text{fc},3} + b_{\text{fc},4}, \tag{26c}$$
$$b_{\text{fc},1} \geq b_{\text{fc},2}, \tag{26d}$$
$$b_{\text{fc},3} \geq b_{\text{fc},4} \tag{26e}$$

To break symmetries in the benchmark problem, we add constraints (27a)–(27m):

$$\neg b^{\text{e}}_{\text{conv},i} \rightarrow C^{\text{e}}_i \leq 4, \qquad \forall i \in [1,2]\,, \tag{27a}$$
$$\neg b^{\text{e}}_{\text{conv},i} \rightarrow S^{\text{e}}_i \leq 1, \qquad \forall i \in [1,2]\,, \tag{27b}$$
$$\neg b^{\text{e}}_{\text{conv},i} \rightarrow P^{\text{e}}_i \leq 0, \qquad \forall i \in [1,2]\,, \tag{27c}$$
$$\neg b^{\text{e}}_{\text{conv},i} \rightarrow F^{\text{e}}_i = 2, \qquad \forall i \in [1,2]\,, \tag{27d}$$
$$\neg b^{\text{e}}_{\text{conv},i} \rightarrow Act^{\text{e}}_i = \text{ReLU}, \qquad \forall i \in [1,2]\,, \tag{27e}$$
$$\neg b_{\text{fc},i} \rightarrow Act^{\text{fc}}_i = \text{ReLU}, \qquad \forall i \in [1,4]\,, \tag{27f}$$
$$\neg b_{\text{fc},1} \rightarrow N^{\text{fc}}_1 \leq 0, \tag{27g}$$
$$\neg b^{\text{d}}_{\text{dec},i} \rightarrow C^{\text{d}}_i \leq 4, \qquad \forall i \in [1,2]\,, \tag{27h}$$
$$\neg b^{\text{d}}_{\text{dec},i} \rightarrow S^{\text{d}}_i \leq 1, \qquad \forall i \in [1,2]\,, \tag{27i}$$
$$\neg b^{\text{d}}_{\text{dec},i} \rightarrow P^{\text{d}}_i \leq 0, \qquad \forall i \in [1,2]\,, \tag{27j}$$
$$\neg b^{\text{d}}_{\text{dec},i} \rightarrow O^{\text{d}}_i \leq 0, \qquad \forall i \in [1,2]\,, \tag{27k}$$
$$\neg b^{\text{d}}_{\text{dec},i} \rightarrow F^{\text{d}}_i = 2, \qquad \forall i \in [1,2]\,, \tag{27l}$$
$$\neg b^{\text{d}}_{\text{dec},1} \rightarrow Act^{\text{d}}_1 = \text{ReLU} \tag{27m}$$

Constraints (27a)–(27m) set layer-specific hyperparameters to pre-defined default values when the associated layer is inactive. We select these defaults as the lower bound for non-categorical variables and the first category for categorical variables. While `SMAC` is unable to handle more complicated constraints restricting outputs of deconvolutional layers, it can handle hierarchical search space structures. For VAE-NAS benchmark runs using `SMAC` as an optimizer we enforce hierarchies according to constraints (27a)–(27m) which deactivate hyperparameters for inactive layers.

Table 4: Hyperparameter names, types, and domains for the VAE-NAS benchmark. The transformation column refers to post-processing computations before passing the hyperparameter value to the neural network training. The architecture with all layers activated comprises `C1-C2-FC1-FC2-L-FC3-FC4-D1-D2`, with `L` referring to the latent space layer.

#	Name	Type	Domain	Transformation
	General			
0	Learning rate	conti.	$[-4.0, -2.0]$	$\alpha = 10^{x_0}$
1	Latent space size	integer	$[16, 64]$	$N^{\text{lat}} = x_1$
2	Num. conv. enc. layers	integer	$[0, 2]$	$N^{\text{e}}_{\text{conv}} = x_2$
3	Num. fully-conn. enc. layers	integer	$[0, 2]$	$N^{\text{e}}_{\text{fc}} = x_3$
4	Num. deconv. dec. layers	integer	$[0, 2]$	$N^{\text{d}}_{\text{dec}} = x_4$
5	Num. fully-conn. dec. layers	integer	$[0, 2]$	$N^{\text{d}}_{\text{fc}} = x_5$
	Encoder			
	Convolutional layer 1 (`C1`)			
6	Number of output channels	integer	$[2, 5]$	$C^{\text{e}}_1 = 2^{x_6}$
7	Stride	integer	$[1, 2]$	$S^{\text{e}}_1 = x_7$
8	Padding	integer	$[0, 3]$	$P^{\text{e}}_1 = x_8$
9	Filter size	categ.	$\{3, 5\}$	$F^{\text{e}}_1 = x_9$
10	Activation function	categ.	{ReLU, PReLU, Leaky ReLU}	$Act^{\text{e}}_1 = x_{10}$
	Convolutional layer 2 (`C2`)			
11	Number of output channels	integer	$[3, 6]$	$C^{\text{e}}_2 = 2^{x_{11}}$
12	Stride	integer	$[1, 2]$	$S^{\text{e}}_2 = x_{12}$
13	Padding	integer	$[0, 3]$	$P^{\text{e}}_2 = x_{13}$
14	Filter size	categ.	$\{3, 5\}$	$F^{\text{e}}_2 = x_{14}$
15	Activation function	categ.	{ReLU, PReLU, Leaky ReLU}	$Act^{\text{e}}_2 = x_{15}$
	Fully-connected layer 1 (`FC1`)			
16	Number of nodes	integer	$[0, 15]$	$N^{\text{fc}}_1 = 64 \times x_{16}$
17	Activation function	categ.	{ReLU, PReLU, Leaky ReLU}	$Act^{\text{fc}}_1 = x_{17}$
	Fully-connected layer 2 (`FC2`)			
18	Activation function	categ.	{ReLU, PReLU, Leaky ReLU}	$Act^{\text{fc}}_2 = x_{18}$
	Decoder			
	Fully-connected layer 3 (`FC3`)			
19	Activation function	categ.	{ReLU, PReLU, Leaky ReLU}	$Act^{\text{fc}}_3 = x_{19}$
	Fully-connected layer 4 (`FC4`)			
20	Activation function	categ.	{ReLU, PReLU, Leaky ReLU}	$Act^{\text{fc}}_4 = x_{20}$
	Deconvolutional layer 1 (`D1`)			
21	Number of input channels	integer	$[3, 6]$	$C^{\text{d}}_1 = 2^{x_{21}}$
22	Stride	integer	$[1, 2]$	$S^{\text{d}}_1 = x_{22}$
23	Padding	integer	$[0, 3]$	$P^{\text{d}}_1 = x_{23}$
24	Output Padding	integer	$[0, 1]$	$O^{\text{d}}_1 = x_{24}$
25	Filter size	categ.	$\{3, 5\}$	$F^{\text{d}}_1 = x_{25}$
26	Activation function	categ.	{ReLU, PReLU, Leaky ReLU}	$Act^{\text{d}}_1 = x_{26}$
	Deconvolutional layer 2 (`D2`)			
27	Number of input channels	integer	$[2, 5]$	$C^{\text{d}}_2 = 2^{x_{27}}$
28	Stride	integer	$[1, 2]$	$S^{\text{d}}_2 = x_{28}$
29	Padding	integer	$[0, 3]$	$P^{\text{d}}_2 = x_{29}$
30	Output Padding	integer	$[0, 1]$	$O^{\text{d}}_2 = x_{30}$
31	Filter size	categ.	$\{3, 5\}$	$F^{\text{d}}_2 = x_{31}$

References

[1] F. Alizadeh and D. Goldfarb. Second-order cone programming. *Mathematical programming*, 95(1):3–51, 2003.

[2] M. Balandat, B. Karrer, D. Jiang, S. Daulton, B. Letham, A. G. Wilson, and E. Bakshy. Botorch: a framework for efficient Monte-Carlo Bayesian optimization. *Advances in Neural Information Processing Systems*, 33:21524–21538, 2020.

[3] R. Baptista and M. Poloczek. Bayesian optimization of combinatorial structures. In *International Conference on Machine Learning*, pages 462–471. PMLR, 2018.

[4] H. Y. Benson and Ü. Sağlam. Mixed-integer second-order cone programming: A survey. In *Theory Driven by Influential Applications*, pages 13–36. INFORMS, 2013.

[5] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. *Advances in Neural Information Processing Systems*, 24, 2011.

[6] D. E. Bernal, Q. Chen, F. Gong, and I. E. Grossmann. Mixed-integer nonlinear decomposition toolbox for Pyomo (MindtPy). In *Computer Aided Chemical Engineering*, volume 44, pages 895–900. Elsevier, 2018.

[7] J. Blank and K. Deb. pymoo: Multi-objective optimization in python. *IEEE Access*, 8:89497–89509, 2020.

[8] P. Bonami, L. T. Biegler, A. R. Conn, G. Cornuéjols, I. E. Grossmann, C. D. Laird, J. Lee, A. Lodi, F. Margot, N. Sawaya, and A. Wächter. An algorithmic framework for convex mixed integer nonlinear programs. *Discrete Optimization*, 5(2):186–204, 2008.

[9] F. Boukouvala and M. G. Ierapetritou. Derivative-free optimization for expensive constrained problems using a novel expected improvement objective function. *AIChE Journal*, 60(7): 2462–2474, 2014.

[10] P. Buathong, D. Ginsbourger, and T. Krityakierne. Kernels over sets of finite sets using RKHS embeddings, with application to Bayesian (combinatorial) optimization. In *International Conference on Artificial Intelligence and Statistics*, pages 2731–2741. PMLR, 2020.

[11] F. Ceccon, J. Jalving, J. Haddad, A. Thebelt, C. Tsay, C. D. Laird, and R. Misener. OMLT: Optimization & machine learning toolkit. *arXiv*, 2202.02414, 2022.

[12] C. A. Coello and E. M. Montes. Constraint-handling in genetic algorithms through the use of dominance-based tournament selection. *Advanced Engineering Informatics*, 16(3):193–203, 2002.

[13] C. Coey, M. Lubin, and J. P. Vielma. Outer approximation with conic certificates for mixed-integer convex problems. *Mathematical Programming Computation*, 12(2):249–293, 2020.

[14] D. D. Cox and S. John. A statistical method for global optimization. In *Proceedings 1992 IEEE International Conference on Systems, Man, and Cybernetics*, pages 1241–1246. IEEE, 1992.

[15] A. Davies and Z. Ghahramani. The random forest kernel and other kernels for big data from random partitions. *arXiv*, 1402.4293, 2014.

[16] E. Daxberger, A. Makarova, M. Turchetta, and A. Krause. Mixed-variable Bayesian optimization. In *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, pages 2633–2639, 2021.

[17] L. Deng. The MNIST database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[18] A. Deshwal, S. Belakaria, and J. R. Doppa. Scalable combinatorial Bayesian optimization with tractable statistical models. *arXiv*, 2008.08177, 2020.

[19] A. Deshwal, S. Belakaria, and J. R. Doppa. Bayesian optimization over hybrid spaces. In *International Conference on Machine Learning*, pages 2632–2643. PMLR, 2021.

[20] S. Drewes and S. Ulbrich. *Mixed integer second order cone programming*. Verlag Dr. Hut Germany, 2009.

[21] T. Elsken, J. H. Metzen, and F. Hutter. Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1):1997–2017, 2019.

[22] P. I. Frazier. A tutorial on Bayesian optimization. *arXiv*, 1807.02811, 2018.

[23] É. Fromont, H. Blockeel, and J. Struyf. Integrating decision tree learning into inductive databases. In *International Workshop on Knowledge Discovery in Inductive Databases*, pages 81–96. Springer, 2006.

[24] J. R. Gardner, G. Pleiss, D. Bindel, K. Q. Weinberger, and A. G. Wilson. GPyTorch: Blackbox matrix-matrix Gaussian process inference with GPU acceleration. In *Advances in Neural Information Processing Systems*, 2018.

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

[26] S. Gopakumar, S. Gupta, S. Rana, V. Nguyen, and S. Venkatesh. Algorithmic assurance: An active approach to algorithmic testing using Bayesian optimisation. *Advances in Neural Information Processing Systems*, 31, 2018.

[27] S. Greenhill, S. Rana, S. Gupta, P. Vellanki, and S. Venkatesh. Bayesian optimization for adaptive experimental design: A review. *IEEE access*, 8:13937–13948, 2020.

[28] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2022.

[29] E. Han, I. Arora, and J. Scarlett. High-dimensional Bayesian optimization via tree-structured additive models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 7630–7638, 2021.

[30] F. Häse, M. Aldeghi, R. J. Hickman, L. M. Roch, and A. Aspuru-Guzik. Gryffin: An algorithm for Bayesian optimization of categorical variables informed by expert knowledge. *Applied Physics Reviews*, 8(3):031406, 2021.

[31] T. Head, MechCoder, G. Louppe, I. Shcherbatyi, fcharras, Z. Vinícius, cmmalone, C. Schröder, nel215, and N. C. et al. scikit-optimize/scikit-optimize: v0.5.2. *Zenodo*, Mar. 2018. doi: 10.5281/zenodo.1207017.

[32] A. Hedar. Test function web pages. Retrieved May 18, 2022, from `http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO.htm`.

[33] H. Hijazi, P. Bonami, and A. Ouorou. An outer-inner approximation for separable mixed-integer nonlinear programs. *INFORMS Journal on Computing*, 26(1):31–44, 2014.

[34] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential Model-Based Optimization for General Algorithm Configuration. In *Proceedings of the 5th International Conference on Learning and Intelligent Optimization*, pages 507–523. Springer-Verlag, 2011.

[35] R. Jenatton, C. Archambeau, J. González, and M. Seeger. Bayesian optimization with tree-structured dependencies. In *International Conference on Machine Learning*, pages 1655–1664. PMLR, 2017.

[36] Y. Jin and J. Branke. Evolutionary optimization in uncertain environments-a survey. *IEEE Transactions on Evolutionary Computation*, 9(3):303–317, 2005.

[37] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T. Liu. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 3149–3157. Curran Associates Inc., 2017.

[38] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv*, 1412.6980, 2014.

[39] D. P. Kingma and M. Welling. Auto-encoding variational Bayes. *arXiv*, 1312.6114, 2013.

[40] A. Krizhevsky, V. Nair, and G. Hinton. CIFAR-10 (Canadian Institute for Advanced Research). URL http://www.cs.toronto.edu/~kriz/cifar.html.

[41] Y.-J. Kuo and H. D. Mittelmann. Interior point methods for second-order cone programming and or applications. *Computational Optimization and Applications*, 28(3):255–285, 2004.

[42] D. Lee, H. Park, and C. D. Yoo. Face alignment using cascade Gaussian process regression trees. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4204–4212, 2015.

[43] J.-C. Lévesque, A. Durand, C. Gagné, and R. Sabourin. Bayesian optimization for conditional hyperparameter spaces. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 286–293. IEEE, 2017.

[44] M. Lindauer, K. Eggensperger, M. Feurer, A. Biedenkapp, D. Deng, C. Benjamins, T. Ruhkopf, R. Sass, and F. Hutter. SMAC3: A Versatile Bayesian Optimization Package for Hyperparameter Optimization. *The Journal of Machine Learning Research*, 23:54–1, 2022.

[45] M. S. Lobo, L. Vandenberghe, S. Boyd, and H. Lebret. Applications of second-order cone programming. *Linear algebra and its applications*, 284(1-3):193–228, 1998.

[46] M. Lubin, E. Yamangil, R. Bent, and J. P. Vielma. Extended formulations in mixed-integer convex programming. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 102–113. Springer, 2016.

[47] A. Lundell and J. Kronqvist. Polyhedral approximation strategies for nonconvex mixed-integer nonlinear programming in SHOT. *Journal of Global Optimization*, 82(4):863–896, 2022.

[48] X. Ma and M. Blaschko. Additive tree-structured covariance function for conditional parameter spaces in Bayesian optimization. In *International Conference on Artificial Intelligence and Statistics*, pages 1015–1025. PMLR, 2020.

[49] V. V. Mišić. Optimization of tree ensembles. *Operations Research*, 68(5):1605–1624, 2020.

[50] M. Mistry, D. Letsios, G. Krennrich, R. M. Lee, and R. Misener. Mixed-integer convex nonlinear optimization with gradient-boosted trees embedded. *INFORMS Journal on Computing*, 33(3): 1103–1119, 2021.

[51] G. Nanfack, P. Temple, and B. Frénay. Constraint enforcement on decision trees: A survey. *ACM Computing Surveys (CSUR)*, 2022.

[52] D. Nguyen, S. Gupta, S. Rana, A. Shilton, and S. Venkatesh. Bayesian optimization for categorical and category-specific continuous inputs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 5256–5263, 2020.

[53] S. Nijssen and E. Fromont. Mining optimal decision trees from itemset lattices. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 530–539, 2007.

[54] T. P. Papalexopoulos, C. Tjandraatmadja, R. Anderson, J. P. Vielma, and D. Belanger. Constrained discrete black-box optimization using mixed-integer programming. In *International Conference on Machine Learning*, pages 17295–17322. PMLR, 2022.

[55] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, and L. e. a. Antiga. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32, pages 8024–8035. Curran Associates, Inc., 2019.

[56] A. Rahimi and B. Recht. Random features for large-scale kernel machines. *Advances in Neural Information Processing Systems*, 20, 2007.

[57] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

[58] S. R. Rath. Convolutional Variational Autoencoder in PyTorch on MNIST Dataset. Retrieved May 18, 2022, from https://debuggercafe.com/convolutional-variational-autoencoder-in-pytorch-on-mnist-dataset/.

[59] B. Ru, A. Alvi, V. Nguyen, M. A. Osborne, and S. Roberts. Bayesian optimisation over multiple continuous and categorical inputs. In *International Conference on Machine Learning*, pages 8276–8285. PMLR, 2020.

[60] R. Sauer, A. Colville, and C. Burwick. Computer points way to more profits. *Hydrocarbon Processing*, 84(2), 1964.

[61] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas. Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.

[62] solab ntu. opt-prob-collection. Retrieved May 18, 2022, from http://www.github.com/solab-ntu/opt-prob-collect/.

[63] S. Surjanovic and D. Bingham. Virtual library of simulation experiments: Test functions and datasets. Retrieved May 18, 2022, from http://www.sfu.ca/~ssurjano.

[64] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: the Condor experience. *Concurrency - Practice and Experience*, 17(2-4):323–356, 2005.

[65] A. Thebelt, J. Kronqvist, R. M. Lee, N. Sudermann-Merx, and R. Misener. Global optimization with ensemble machine learning models. In *Computer Aided Chemical Engineering*, volume 48, pages 1981–1986. Elsevier, 2020.

[66] A. Thebelt, J. Kronqvist, M. Mistry, R. M. Lee, N. Sudermann-Merx, and R. Misener. ENTMOOT: A framework for optimization over ensemble tree models. *Computers & Chemical Engineering*, 151:107343, 2021.

[67] A. Thebelt, C. Tsay, R. M. Lee, N. Sudermann-Merx, D. Walz, T. Tranter, and R. Misener. Multi-objective constrained optimization for energy applications via tree ensembles. *Applied Energy*, 306:118061, 2022.

[68] A. Thebelt, J. Wiebe, J. Kronqvist, C. Tsay, and R. Misener. Maximizing information from chemical engineering data sets: Applications to machine learning. *Chemical Engineering Science*, 252:117469, 2022.

[69] A. Tran, J. Sun, J. M. Furlan, K. V. Pagalthivarthi, R. J. Visintainer, and Y. Wang. pBO-2GP-3B: A batch parallel known/unknown constrained Bayesian optimization with feasibility classification and its applications in computational fluid dynamics. *Computer Methods in Applied Mechanics and Engineering*, 347:827–852, 2019.

[70] M. Trencseni. Solving CIFAR-10 with Pytorch and SKL. Retrieved May 18, 2022, from https://bytepawn.com/solving-cifar-10-with-pytorch-and-skl.html.

[71] J. P. Vielma, I. Dunning, J. Huchette, and M. Lubin. Extended formulations in mixed integer conic quadratic programming. *Mathematical Programming Computation*, 9(3):369–418, 2017.

[72] C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter. Nas-bench-101: Towards reproducible neural architecture search. In *International Conference on Machine Learning*, pages 7105–7114. PMLR, 2019.

[73] A. Zafari, R. Zurita-Milla, and E. Izquierdo-Verdiguier. Evaluating the performance of a random forest kernel for land cover classification. *Remote sensing*, 11(5):575, 2019.