A INFERENC

A.1 NUTS

We use the NUTS sampler implemented in NumPyro [Phan et al., 2019; Bingham et al., 2019], which leverages JAX for efficient hardware acceleration [Bradbury et al., 2020]. In most of our experiments (see Sec. E for exceptions) we run NUTS for $768 = 512 + 256$ steps where the first $N_{\text{warmup}} = 512$ samples are for burn-in and (diagonal) mass matrix adaptation (and thus discarded), and where we retain every 16th sample among the final $N_{\text{post}} = 256$ samples (i.e. sample thinning), yielding a total of $L = 16$ approximate posterior samples. It is these $L$ samples that are then used to compute Eqns. (4), (5), (10). We also limit the maximum tree depth in NUTS to 6.

We note that these choices are somewhat conservative, and in many settings we would expect good results with fewer samples. Indeed on the Branin test function, see Fig. 6, we find a relatively marginal drop in performance when we reduce the NUTS sampling budget as follows: i) reduce the number of warmup samples from 512 to 128; ii) reduce the number of post-warmup samples from 256 to 128; and iii) reduce the total number of retained samples from 16 to 8. We expect broadly similar results for many other problems. See Sec. C for corresponding runtime results.

We note that these choices are somewhat conservative, and in many settings we would expect good results with fewer samples. Indeed on the Branin test function, see Fig. 6, we find a relatively marginal drop in performance when we reduce the NUTS sampling budget as follows: i) reduce the number of warmup samples from 512 to 128; ii) reduce the number of post-warmup samples from 256 to 128; and iii) reduce the total number of retained samples from 16 to 8. We expect broadly similar results for many other problems. See Sec. C for corresponding runtime results.

A.2 MAP

We run the Adam optimizer [Kingma and Ba, 2015] for 1500 steps and with a learning rate of 0.02 and $\beta_1 = 0.50$ to maximize the log density

$$U_s(\psi_s|\tau_s) = \log p(y|X, \psi_s) + \log p(\psi_s|\tau_s)$$

w.r.t. $\psi_s$ for $S = 4$ pre-selected values of $\tau_s$: $\tau_s \in \{1, 10^{-1}, 10^{-2}, 10^{-3}\}$. This optimization is trivially optimized across $S$.

For each $s = 1, \ldots, S$ we then compute the leave-one-out predictive log likelihood using the mean and variance functions given in Eqns. (4)-(5). We then choose the value of $s$ that maximizes this predictive log likelihood and use the corresponding kernel hyperparameter $\psi_s$ to compute the expected improvement in Eqn. (10).

Supplement for the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence (UAI 2021).
Figure 6: We depict how SAASBO-NUTS performs on Branin as we reduce the sampling budget \((N_{\text{warmup}}, N_{\text{post}}, L)\) from \((512, 256, 16)\) to \((N_{\text{warmup}}, N_{\text{post}}, L) = (128, 128, 8)\). We compare performance w.r.t. the best minimum found (the mean is depicted by a thick line and shaded bands denote standard errors). Each curve corresponds to 60 independent replications of Algorithm 1.

### A.3 NO DISCRETE LATENT VARIABLES

As discussed briefly in the main text, it is important that the SAAS prior defined in Sec. 4.1 does not include any discrete latent variables. Indeed a natural alternative to our model would introduce \(D\) binary-valued latent variables that control whether or not a given dimension is relevant to modeling \(f_{\text{obj}}\). However, inference in any such model can be very challenging, as it requires exploring an extremely large discrete space of size \(2^D\). Our model can be understood as a continuous relaxation of such an approach. This is a significant advantage since it means we can leverage gradient information to efficiently explore the posterior. Indeed, the structure of our sparsity-inducing prior closely mirrors the justly famous Horseshoe prior [Carvalho et al., 2009], which is a popular prior for Sparse Bayesian linear regression. We note that in contrast to the linear regression setting of the Horseshoe prior, our sparsity-inducing prior governs inverse squared length scales in a non-linear kernel and not variances. While we expect that any prior that concentrates \(\rho_i\) at zero can exhibit good empirical performance in the setting of high-dimensional BO, this raises the important question whether distributional assumptions other than those in Eqn. (8) may be better suited to governing our prior expectations about \(\rho_i\). Making a careful investigation of this point is an interesting direction for future work.

### B EXPECTED IMPROVEMENT MAXIMIZATION

We first form a scrambled Sobol sequence \(x_{1:Q}\) (see e.g. [Owen, 2003]) of length \(Q = 5000\) in the \(D\)-dimensional domain \(D\). We then compute the expected improvement in Eqn. (10) in parallel for each point in the Sobol sequence. We then choose the top \(K = 3\) points in \(x_{1:Q}\) that yield the largest EIs. For each of these \(K\) approximate maximizers we run L-BFGS [Zhu et al., 1997] initialized with the approximate maximizer and using the implementation provided by Scipy (in particular \texttt{fmin_l_bfgs_b}) to obtain the final query point \(x_{\text{next}}\), which (approximately) maximizes Eqn. (10). We limit \texttt{fmin_l_bfgs_b} to use a maximum of 100 function evaluations.

### C RUNTIME EXPERIMENT

We measure the runtime of SAASBO as well as each baseline method on the Branin test problem. See Table 1 for the results. We record runtimes for both the default SAASBO-NUTS settings described in Sec. A.1 as well as one with a reduced NUTS sampling budget. While SAASBO requires more time per iteration than other methods such as TuRBBO and HeSBO, the overhead is relatively moderate in the setting where the black-box function \(f_{\text{obj}}\) is very expensive to evaluate. We note that after reducing the NUTS sampling budget to \((N_{\text{warmup}}, N_{\text{post}}, L) = (128, 128, 8)\) about 75% of the runtime is devoted to EI optimization. Since our current implementation executes \(K = 3\) runs of L-BFGS serially, this runtime could be reduced further by executing L-BFGS in parallel.

<table>
<thead>
<tr>
<th>Method</th>
<th>Time / iteration</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAASBO (default)</td>
<td>26.51 seconds</td>
</tr>
<tr>
<td>SAASBO (128-128-8)</td>
<td>19.21 seconds</td>
</tr>
<tr>
<td>TuRBBO</td>
<td>1.52 seconds</td>
</tr>
<tr>
<td>SMAC</td>
<td>12.12 seconds</td>
</tr>
<tr>
<td>EBO</td>
<td>128.10 seconds</td>
</tr>
<tr>
<td>ALEBO ((d_e = 5))</td>
<td>4.34 seconds</td>
</tr>
<tr>
<td>ALEBO ((d_e = 10))</td>
<td>11.91 seconds</td>
</tr>
<tr>
<td>HeSBO ((d_e = 5))</td>
<td>0.70 seconds</td>
</tr>
<tr>
<td>HeSBO ((d_e = 10))</td>
<td>1.51 seconds</td>
</tr>
<tr>
<td>CMA-ES</td>
<td>&lt; 0.1 seconds</td>
</tr>
<tr>
<td>Sobol</td>
<td>&lt; 0.01 seconds</td>
</tr>
</tbody>
</table>
D ADDITIONAL FIGURES AND EXPERIMENTS

D.1 MODEL FITTING

In Fig. 7 we reproduce the experiment described in Sec. 5.1 with the difference that we replace the RBF kernel with a Matérn-5/2 kernel.

We note that the qualitative behavior in Fig. 7 matches the behavior in Fig. 1. In particular, only the sparsity-inducing SAAS function prior provides a good fit. This emphasizes that the potential for drastic overfitting that arises when fitting a non-sparse GP in high dimensions is fundamental and is not ameliorated by using a different kernel. In particular the fact that the Matérn-5/2 kernel decays less rapidly at large distances as compared to the RBF kernel (quadratically instead of exponentially) does not prevent the non-sparse models from yielding essentially trivial predictions across most of the domain D.

D.2 SVM RELEVANCE PLOTS

In Fig. 8 we explore the relevant subspace identified by SAASBO during the course of optimization of the SVM problem discussed in Sec. 5.6. We see that the three most important hyperparameters, namely the regularization hyperparameters, are consistently found more or less immediately once the initial Sobol phase of Algorithm 1 is over. This explains the rapid early progress that SAASBO makes in Fig. 4 during optimization. We note that the 4th most relevant dimension turns out to be a length scale for a patient ID feature, which makes sense given the importance of this feature to the regression problem.

D.3 SVM ABLATION STUDY

In Fig. 9 we depict results from an ablation study of SAASBO in the context of the SVM problem. First, as a companion to Fig. 4 and Fig. 7, we compare the BO performance of the SAAS function prior to a non-sparse function prior that places weak priors on the length scales. As we would expect from Fig. 4 and Fig. 7 the resulting BO performance is very poor for the non-sparse prior. Second, we also compare the default RBF kernel to a Matérn-5/2 kernel. We find that, at least on this problem, both kernels lead to similar BO performance.

D.4 ROTATED HARTMANN

In this experiment we study whether the axis-aligned assumption in SAAS leads to degraded performance on non-axis-aligned objective functions. In particular, we consider the Hartmann function \( f_{\text{hart}} \) for \( d = 6 \) embedded in \( D = 100 \) dimensions. Given a projection dimensionality \( d_p \geq d \), we generate a random linear projection \( P_{d_p} \in \mathbb{R}^{d \times d_p} \) where \( (P_{d_p})_{ij} \sim \mathcal{N}(0, 1/d_p) \). The task is to optimize \( \tilde{f}(x) = f_{\text{hart}}(P_{d_p} x, 1:d_p - z) \) where \( x \in [0, 1]^D \)
E.1 MODEL FIT EXPERIMENT

In the model fit experiment in Sec. 5.1 we take data collected from two different runs of SAASBO in $D = 100$. We use one run as training data and the second run as test data, each with $N = 100$ datapoints. To construct datasets in $D = 30$ dimensions we include the 6 relevant dimensions as well as 24 randomly chosen redundant dimensions and drop all remaining dimensions.

E.2 INFERENCE AND HYPERPARAMETER COMPARISON EXPERIMENT

For the experiment in Sec. 5.2 that is depicted in Fig. 2 we initialize SAASBO with $m = 10$ points from a Sobol sequence.

E.3 BASELINES

We compare SAASBO to ALEBO, CMA-ES, EBO, HeSBO, SMAC, Sobol, and TuRBO. For ALEBO and HeSBO we use the implementations in BoTorch [Balandat et al., 2020] with the same settings that were used by [Letham et al., 2020]. We consider embeddings of dimensionality $d_e = 5$ and $d_e = 10$ on the synthetic problems, which is similar to the $d_e = d$ and $d_e = 2d$ heuristics that were considered in [Nayebi et al., 2019] as well as [Letham et al., 2020]. As the true active dimensionality $d$ of $f_{obj}$ is unknown, we do not allow any method to explicitly use this additional information. For the three real-world experiments, $d_e = 5$ does not work well on any problem so we instead report results for $d_e = 10$ and $d_e = 20$.

For CMA-ES we use the pycma$^4$ implementation. CMA-ES is initialized using a random point in the domain and uses the default initial step-size of 0.25. Recall that the domain is normalized to $[0, 1]^D$ for all problems. We run EBO using the reference implementation by the author$^2$ with the default settings. EBO requires knowing the value of the function at the global optimum. Similarly to [Letham et al., 2020] we provide this value to EBO for all problems, but note that EBO still performs poorly on all problems apart from Brainin and SVM.

Our comparison to SMAC uses SMAC4HPO, which is implemented in SMAC$^7$. On all problems we run SMAC in deterministic mode, as all problems considered in this paper are noise-free. For Sobol we use the SobolEngine implementation in PyTorch. Finally, we compare to TuRBO with a single trust region due to the limited evaluation budget; we use the implementation provided by the author$^3$.

https://github.com/SMAC4HPO
https://github.com/zi-w/Ensemble-Bayesian-Optimization
https://github.com/autoki/SMAC3
https://github.com/uber-research/TuRBO
We consider the rover trajectory optimization problem that was also considered in Wang et al. [2018]. The goal is to optimize the trajectory of a rover where this trajectory is determined by fitting a B-spline to 30 waypoints in the 2D plane. While the original problem had a pre-determined origin and destination, the resulting B-spline was not constrained to start and end at these positions. To make the problem easier, we force the B-spline to start and end at these pre-determined positions. Additionally, we use 50 waypoints points, which results in a 100-dimensional optimization problem. The reward function for the trajectory is computed in the same way as in Wang et al. [2018], namely we integrate over the trajectory penalizing collisions with potential objects. On this problem we initialize all methods except for CMA-ES with \( m = 20 \) initial points.

### E.4 SYNTHETIC PROBLEMS

We consider three standard synthetic functions from the optimization literature. Branin is a 2-dimensional function that we embed in a 100-dimensional space. We consider the standard domain \([-5, 10] \times [0, 15]\) before normalizing the domain to \([0, 1]^{100}\). For Hartmann, we consider the \( d = 6 \) version on the domain \([0, 1]^6\) before embedding it in a 100-dimensional space. For Rosenbrock, we use \( d = 3 \) and the domain \([-2, 2]^3\), which we then embed and normalize so that the full domain is \([0, 1]^{100}\). Rosenbrock is a function that is challenging to model, as there are large function values at the boundary of the domain. For this reason all methods minimize \( \log(1 + f_{\text{obj}}(x)) \). All methods except for CMA-ES are initialized with \( m = 10 \) initial points for Branin and Rosenbrock and \( m = 20 \) initial points for Hartmann.

### E.5 ROVER

We randomly choose 5000 training and 5000 test points from the 385-dimensional “CT slice” UCI dataset [Dua and Graff, 2019]. We normalize the inputs and scalar output so that e.g. the test RMSE of a trivial zero prediction is given by 1.0. Our domain \( D \) then consists of 385 kernel (log) length scales and 3 regularization hyperparameters for a kernel support vector machine fit with Scikit-learn.
While it is worth emphasizing that there are constrained optimization problems that can quickly exploit structure in $f_{\text{obj}}$ even though there is no obvious low-dimensional structure.

For SAASBO we use the NUTS settings described in Sec. [A1] for $t \leq 150$. To lower the runtime after iteration $t > 150$ we collect $384 = 192 + 192$ NUTS samples and retain every $24^{th}$ of the final 192 samples, resulting in a total of $L = 8$ retained samples. We note while this may hurt the accuracy of the inferred GP model, SAASBO still performs very well on this problem and outperforms other methods by a large margin. As we consider a larger evaluation budget on this problem we initialize all methods except for CMA-ES with $m = 20$ initial points.

### E.7 MOPTA VEHICLE DESIGN

We consider the vehicle design problem MOPTA08 which is a challenging 124-dimensional real-world high-dimensional BO problem [Jones 2008]. The goal in this problem is to minimize the mass of a vehicle subject to 68 performance constraints. The $D = 124$ design variables describe materials, gauges, and vehicle shape. While this problem is originally formulated as a constrained optimization problem, we make it unconstrained by converting the constraints into a soft constraint. In particular, we consider minimizing $f_{\text{obj}}(x) + 10 \sum_{i=1}^{68} \max(0, c_i(x))$ where the 68 constraints are of the form $c_i(x) \leq 0$. This penalty is chosen to be small enough to have most of the signal come from $f_{\text{obj}}$ while at the same time discouraging large constraint violations. While it is worth emphasizing that there are constrained optimization methods that can explicitly handle the constraint, this problem shows that SAASBO can quickly exploit structure in $f_{\text{obj}}$ even though there is no obvious low-dimensional structure.

For SAASBO we use the NUTS settings described in Sec. [A1] for $t \leq 150$. To lower the runtime after iteration $t > 150$ we collect $384 = 192 + 192$ NUTS samples and retain every $24^{th}$ of the final 192 samples, resulting in a total of $L = 8$ retained samples. We note while this may hurt the accuracy of the inferred GP model, SAASBO still performs very well on this problem and outperforms other methods by a large margin. As we consider a larger evaluation budget on this problem we initialize all methods except for CMA-ES with $m = 50$ initial points.

### References