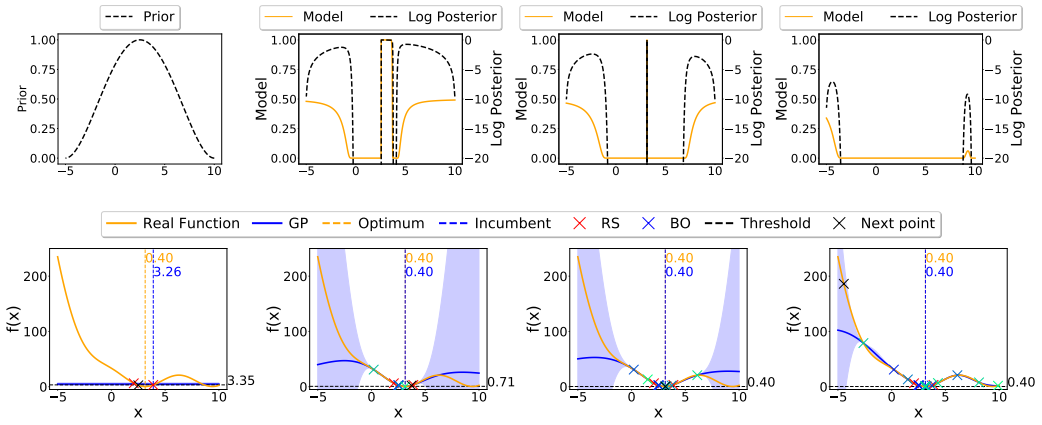


(a) 0 BO iterations (b) 5 BO iterations (c) 10 BO iterations (d) 20 BO iterations

Figure 6: Breakdown of the prior $P_g(\boldsymbol{x}) = \mathcal{B}(3,3)$, the model $M_g(\boldsymbol{x})$, and the pseudo-posterior $g(\boldsymbol{x})$ (top row) for the 1-dimensional Branin function (bottom row) and their evolution over the optimization iterations. Two random points from the prior are sampled to initialize the GP model (bottom row) before starting PrBO. The blue/green crosses denote PrBO samples, with green samples denoting later iterations. The blue line and shaded area show the mean and uncertainty of the GP model.

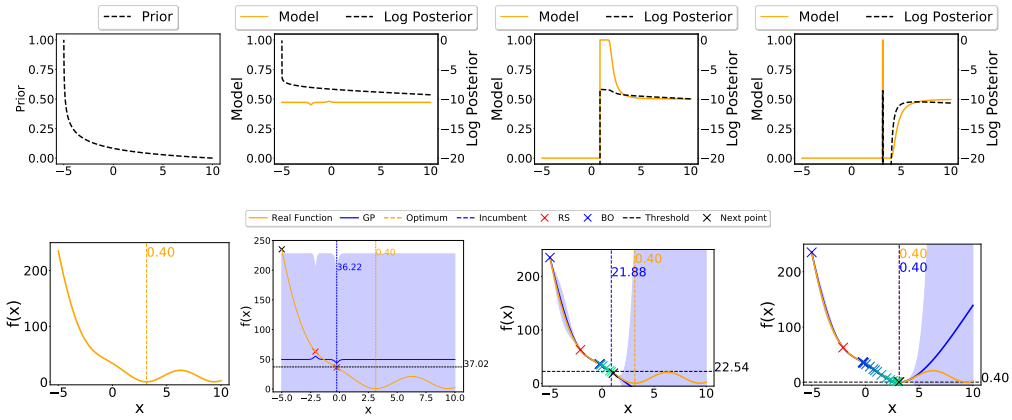
A Model and Pseudo-posterior Visualization

This section visualizes the prior $P_g(\boldsymbol{x})$, the model $M_g(\boldsymbol{x})$, and the pseudo-posterior $g(\boldsymbol{x})$ for a 1-dimensional Branin function and their evolution over the optimization iterations. We define this function by setting the second dimension of the Branin function to the global optimum $x_2 = 2.275$ and optimizing the first dimension. We perform an initial design of $D+1 = 2$ random points sampled from the prior and use a GP as predictive model. We use a Beta distribution prior $P_g(\boldsymbol{x}) = \mathcal{B}(3,3)$ which resembles a truncated Gaussian centered close to the global optimum and compute the model $\mathcal{M}_g(\boldsymbol{x})$ and pseudo-posterior $g(\boldsymbol{x})$ following Eq. (2) and (3) respectively. Figure 6 shows the optimization at different stages.

Figure 6a shows the initialization phase (bottom) and the Beta prior (top). After 5 BO iterations, in Figure 6b (top), the pseudo-posterior is high near the global minimum, around $x = \pi$, where both the prior and the model agree there are good points. After 10 BO iterations in Figure 6c (top), there are three regions with high pseudo-posterior. The middle region, where PrBO is exploiting until the optimum is found, and two regions to the right and left, which will lead to future exploration as shown in Figure 6d (bottom) on the right and left of the global optimum in light green crosses. After 20 iterations, see Figure 6d (top), the pseudo-posterior vanishes where the model is certain there will be no improvement, but it is high wherever there is uncertainty in the GP. Note that the influence of the prior after 20 iterations is weaker, because of t/β in Eq. (3).

B Prior Forgetting Supplementary Experiments

In this section, we show additional evidence that PrBO can recover from wrongly defined priors so to complement section 4.2. Figure 7 shows PrBO on the 1D Branin function as in Figure 2 but with a decay prior. Column (a) of Figure 7 shows the decay prior and the 1D Branin function. This prior emphasizes the wrong belief that the optimum is likely located on the left side around $\mathrm{x} = -5$ while the optimum is located at the orange dashed line. Columns (b), (c), and (d) of Figure 7 show PrBO on the 1D Branin after $D + 1 = 2$ initial samples and 0, 10, and 20 BO iterations, respectively. In the beginning of BO, as shown in column (b), the pseudo-posterior is nearly identical to the prior and guides PrBO towards the left region of the space. As more points are sampled, the model becomes more accurate and starts guiding the pseudo-posterior away from the wrong prior (column (c)). Notably, the pseudo-posterior before $\mathrm{x} = 0$ falls to 0, as the predictive model is certain there will



(a) No samples (b) 0 BO iterations (c) 10 BO iterations (d) 20 BO iterations

Figure 7: PrBO on the 1D Branin function with a decay prior. The leftmost column shows the log pseudo-posterior before any samples are evaluated, in this case, the pseudo-posterior is equal to the decay prior. The other columns show the model and pseudo-posterior after 0 (only random samples), 10, and 20 BO iterations. 2 random samples are used to initialize the GP model.

be no improvement from sampling this region. After 20 iterations, PrBO finds the optimum region, despite the poor start (column (d)). The peak in the pseudo-posterior in column (d) shows PrBO will continue to exploit the optimum region as it is not certain if the exact optimum has been found. The pseudo-posterior is also high in the high uncertainty region after $x = 4$, showing PrBO will explore that region after it finds the optimum.

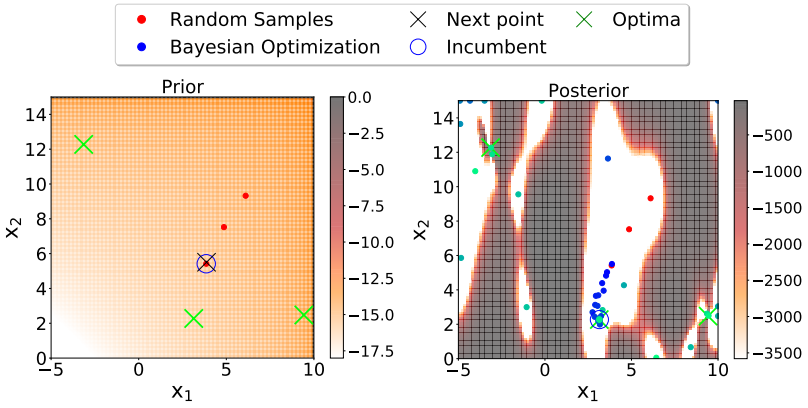


Figure 8: PrBO on the Branin function with exponential priors for both dimensions. (a) shows the log pseudo-posterior before any samples are evaluated, in this case, the pseudo-posterior is equal to the prior; the green crosses are the optima. (b) shows the result of optimization after 3 initialization samples drawn from the prior at random and 50 BO iterations. The dots in (b) show the points explored by PrBO, with greener points denoting later iterations. The colored heatmap shows the log of the pseudo-posterior $g(\boldsymbol{x})$.

Figure 8 shows PrBO on the standard 2D Branin function. We use exponential priors for both dimensions, which guides optimization towards a region with only poor performing high function values. 8a shows the prior and 8b shows optimization results after $D + 1 = 3$ initialization samples and 50 BO iterations. Note that, once again, optimization begins near the region incentivized by the prior, but moves away from the prior and towards the optima as BO progresses. After 50 BO iterations, PrBO finds all three optima regions of the Branin.

C MATHEMATICAL DERIVATIONS

C.1 EI DERIVATION

Here, we provide a full derivation of Eq. (5):

$$EI_{f_\gamma}(\boldsymbol{x}) := \int_{-\inf}^{\inf} \max(f_\gamma - y, 0) p(y|\boldsymbol{x}) dy = \int_{-\inf}^{f_\gamma} (f_\gamma - y) \frac{p(\boldsymbol{x}|y)p(y)}{p(\boldsymbol{x})} dy. \tag{6}$$

As defined in Section 3.2, $p(y < f_\gamma) = \gamma$ and γ is a quantile of the observed objective values $\{y^{(i)}\}$. Then $p(\boldsymbol{x}) = \int_{\mathbb{R}} p(\boldsymbol{x}|y)p(y)dy = \gamma g(\boldsymbol{x}) + (1-\gamma)b(\boldsymbol{x})$, where $g(\boldsymbol{x})$ and $b(\boldsymbol{x})$ are the posteriors introduced in Section 3.3. Therefore

$$\int_{-\inf}^{f_\gamma} (f_\gamma - y) p(\boldsymbol{x}|y)p(y)dy = g(\boldsymbol{x}) \int_{-\inf}^{f_\gamma} (f_\gamma - y)p(y)dy = \gamma f_\gamma g(\boldsymbol{x}) - g(\boldsymbol{x}) \int_{-\inf}^{f_\gamma} yp(y)dy, \tag{7}$$

so that finally

$$EI_{f_\gamma}(\boldsymbol{x}) = \frac{\gamma f_\gamma g(\boldsymbol{x}) - g(\boldsymbol{x}) \int_{-\inf}^{f_\gamma} yp(y)dy}{\gamma g(\boldsymbol{x}) + (1-\gamma)b(\boldsymbol{x})} \propto \left(\gamma + \frac{b(\boldsymbol{x})}{g(\boldsymbol{x})}(1-\gamma)\right)^{-1}. \tag{8}$$

C.2 PROOF OF PROPOSITION 1

Here, we provide the proof of Proposition 1:

$$\begin{aligned}
\lim_{t\to\infty} \operatorname*{arg\,max}_{\boldsymbol{x}\in\mathcal{X}} EI_{f_\gamma}(\boldsymbol{x}) &= \lim_{t\to\infty} \operatorname*{arg\,max}_{\boldsymbol{x}\in\mathcal{X}} \int_{-\inf}^{f_\gamma} (f_\gamma - y)p(\boldsymbol{x}|y)p(y)dy && (9)\\
&= \lim_{t\to\infty} \operatorname*{arg\,max}_{\boldsymbol{x}\in\mathcal{X}} g(\boldsymbol{x}) \int_{-\inf}^{f_\gamma} (f_\gamma - y)p(y)dy && (10)\\
&= \lim_{t\to\infty} \operatorname*{arg\,max}_{\boldsymbol{x}\in\mathcal{X}} \left(\gamma f_\gamma g(\boldsymbol{x}) - g(\boldsymbol{x}) \int_{-\inf}^{f_\gamma} yp(y)dy\right) && (11)\\
&= \lim_{t\to\infty} \operatorname*{arg\,max}_{\boldsymbol{x}\in\mathcal{X}} \frac{\gamma f_\gamma g(\boldsymbol{x}) - g(\boldsymbol{x}) \int_{-\inf}^{f_\gamma} yp(y)dy}{\gamma g(\boldsymbol{x}) + (1-\gamma)b(\boldsymbol{x})} && (12)\\
&= \lim_{t\to\infty} \operatorname*{arg\,max}_{\boldsymbol{x}\in\mathcal{X}} \left(\gamma + \frac{b(\boldsymbol{x})}{g(\boldsymbol{x})}(1-\gamma)\right)^{-1} && (13)\\
&= \lim_{t\to\infty} \operatorname*{arg\,max}_{\boldsymbol{x}\in\mathcal{X}} \left(\gamma + \frac{b(\boldsymbol{x})}{g(\boldsymbol{x})}(1-\gamma)\right)^{-\frac{1}{t}} && (14)\\
&= \lim_{t\to\infty} \operatorname*{arg\,max}_{\boldsymbol{x}\in\mathcal{X}} \left(\gamma + \frac{P_b(\boldsymbol{x})\mathcal{M}_b(\boldsymbol{x})^{\frac{t}{\beta}}}{P_g(\boldsymbol{x})\mathcal{M}_g(\boldsymbol{x})^{\frac{t}{\beta}}}(1-\gamma)\right)^{-\frac{1}{t}} && (15)\\
&= \lim_{t\to\infty} \operatorname*{arg\,max}_{\boldsymbol{x}\in\mathcal{X}} \left(\frac{P_b(\boldsymbol{x})\mathcal{M}_b(\boldsymbol{x})^{\frac{t}{\beta}}}{P_g(\boldsymbol{x})\mathcal{M}_g(\boldsymbol{x})^{\frac{t}{\beta}}}(1-\gamma)\right)^{-\frac{1}{t}} && (16)\\
&= \lim_{t\to\infty} \operatorname*{arg\,max}_{\boldsymbol{x}\in\mathcal{X}} \left(\frac{P_b(\boldsymbol{x})}{P_g(\boldsymbol{x})}\right)^{-\frac{1}{t}} \left(\frac{\mathcal{M}_b(\boldsymbol{x})^{\frac{t}{\beta}}}{\mathcal{M}_g(\boldsymbol{x})^{\frac{t}{\beta}}}\right)^{-\frac{1}{t}} (1-\gamma)^{-\frac{1}{t}} && (17)\\
&= \lim_{t\to\infty} \operatorname*{arg\,max}_{\boldsymbol{x}\in\mathcal{X}} \left(\frac{\mathcal{M}_b(\boldsymbol{x})}{\mathcal{M}_g(\boldsymbol{x})}\right)^{-\frac{1}{\beta}} && (18)\\
&= \lim_{t\to\infty} \operatorname*{arg\,max}_{\boldsymbol{x}\in\mathcal{X}} \left(\frac{1-\mathcal{M}_g(\boldsymbol{x})}{\mathcal{M}_g(\boldsymbol{x})}\right)^{-\frac{1}{\beta}} && (19)
\end{aligned}$$

$$= \lim_{t\to\infty} \arg\max_{\boldsymbol{x}\in\mathcal{X}} \left(\frac{1}{\mathcal{M}_g(\boldsymbol{x})} - 1\right)^{-\frac{1}{\beta}} \tag{20}$$

$$= \lim_{t\to\infty} \arg\max_{\boldsymbol{x}\in\mathcal{X}} \left(\mathcal{M}_g(\boldsymbol{x})\right)^{\frac{1}{\beta}} \tag{21}$$

$$= \lim_{t\to\infty} \arg\max_{\boldsymbol{x}\in\mathcal{X}} \mathcal{M}_g(\boldsymbol{x}) \tag{22}$$

$$\tag{23}$$

This shows that as iterations progress, the model grows more important. If PrBO is run long enough, the prior washes out and PrBO only trusts the probabilistic model.

D EXPERIMENTAL SETUP

We use a combination of publicly available implementations for our predictive models. For our Gaussian Process (GP) model, we use GPy's (GPy, since 2012) GP implementation with the Matérn5/2 kernel. We use different length-scales for each input dimensions, learned via Automatic Relevance Determination (ARD) (Neal, 2012). For our Random Forests (RF), we use scikit-learn's RF implementation (Pedregosa et al., 2011). We set the fraction of features per split to 0.5, the minimum number of samples for a split to 5 and disable bagging. We also adapt our RF implementation to use the same split selection approach as Hutter et al. (2014).

For our constrained Bayesian Optimization (cBO) approach, we use scikit-learn's RF classifier, trained on previously explored configurations, to predict the probability of a configuration being feasible. We then weight our EI acquisition function by this probability of feasibility, as proposed by Gardner et al. (2014). We normalize our EI acquisition function before considering the probability of feasibility, to ensure both values are in the same range. This cBO implementation is used in the Spatial use-case as in Nardi et al. (2019).

For all experiments, we set the model weight hyperparameter to $\beta = 10$ and the model quantile to $\gamma = 0.05$, see Appendices M and L. Before starting the main BO loop, PrBO is initialized by random sampling $D + 1$ points from the prior, where D is the number of input variables. We use the public implementation of Spearmint[3], which by default uses 2 random samples for initialization. We set the bandwidth of our KDE priors to $100n^{-\frac{1}{D}}$, where D is the number of input dimensions, see Appendix K. We normalize our KDE priors before computing the pseudo-posterior, to ensure they are in the same range as our model. We also implement interleaving which randomly samples a point to explore during BO with a 10% chance.

We optimize our EI acquisition function using a multi-start local search, similar to the one used in SMAC (Hutter et al., 2011). Namely, we start local searches on the 10 best points evaluated in previous BO iterations, on the 10 best performing points from a set of $10{,}000$ random samples and on the 10 best performing points from $10{,}000$ random samples drawn from the prior. To compute the neighbors of each of these 30 total points, we normalize the range of each objective to $[0, 1]$ and randomly sample four neighbors from a truncated Gaussian centered at the original value and with standard deviation $\sigma = 0.2$.

We use four synthetic benchmarks in our experiments.

Branin. The Branin function is a well-known synthetic benchmark for optimization problems (Dixon, 1978). The Branin function has two input dimensions and three global minima.

SVM. is a hyperparameter-optimization benchmark in 2D based on Profet (Klein et al., 2019). This benchmark is generated by a generative meta-model built using a set of SVM classification models trained on 16 OpenML tasks. The benchmark has two input parameters, corresponding to SVM hyperparameters.

FC-Net. is a hyperparameter and architecture optimization benchmark in 6D based on Profet. The FC-Net benchmark is generated by a generative meta-model built using a set of feed-forward neural networks trained on the same 16 OpenML tasks as the SVM benchmark. The benchmark has six input parameters corresponding to network hyperparameters.

[3]https://github.com/HIPS/Spearmint

Table 1: Search spaces for our synthetic benchmarks. For the Profet benchmarks, we report the original ranges and whether or not a log scale was used.

Benchmark	Parameter name	Parameter values	Log scale
Branin	x_1	$[-5, 10]$	-
	x_2	$[0, 15]$	-
SVM	C	$[e^{-10}, e^{10}]$	✓
	γ	$[e^{-10}, e^{10}]$	✓
FCNet	learning rate	$[10^{-6}, 10^{-1}]$	✓
	batch size	$[8, 128]$	✓
	units layer 1	$[16, 512]$	✓
	units layer 2	$[16, 512]$	✓
	dropout rate l1	$[0.0, 0.99]$	-
	dropout rate l2	$[0.0, 0.99]$	-
XGBoost	learning rate	$[10^{-6}, 10^{-1}]$	✓
	gamma	$[0, 2]$	-
	L1 regularization	$[10^{-5}, 10^{3}]$	✓
	L2 regularization	$[10^{-5}, 10^{3}]$	✓
	number of estimators	$[10, 500]$	-
	subsampling	$[0.1, 1]$	-
	maximum depth	$[1, 15]$	-
	minimum child weight	$[0, 20]$	-

XGBoost. is hyperparameter-optimization benchmark in 8D based on Profet. The XGBoost benchmark is generated by a generative meta-model built using a set of XGBoost regression models in 11 UCI datasets. The benchmark has eight input parameters, corresponding to XGBoost hyperparameters.

The search spaces for each benchmark are summarized in Table 1. For the Profet benchmarks, we report the original ranges and whether or not a log scale was used. However, in practice, Profet's generative model transforms the range of all hyperparameters to a linear $[0, 1]$ range. We use Emukit's public implementation for these benchmarks (Paleyes et al., 2019).

E SVM Regret Comparison

In addition to the experiments in Section 4.3, we show the performance of PrBO on the SVM benchmark. Figure 9 shows a log regret comparison of PrBO, Spearmint, Prior Sampling and $10{,}000\times$ RS. We note that the results are similar to the other benchmarks in Figure 4. Namely, PrBO with a strong prior outperforms RS and spearmint. PrBO also outperforms Spearmint with a weak prior and even with a uniform prior.

F Spatial Real-world Application

`Spatial` (Koeplinger et al., 2018) is a programming language and corresponding compiler for the design of application accelerators on reconfigurable architectures, e.g. field-programmable gate arrays (FPGAs). These reconfigurable architectures are a type of logic chip that can be reconfigured via software to implement different applications. `Spatial` provides users with a high-level of abstraction for hardware design, so that they can easily design their own applications on FPGAs. It allows users to specify parameters that do not change the behavior of the application, but impact the runtime and resource-usage (e.g. logic units) of the final design. During compilation, the `Spatial` compiler estimates the ranges of these parameters and estimates the resource-usage and runtime of the application for different parameter values. These parameters can then be optimized during compilation in order to achieve the design with the fastest runtime. We fully integrate PrBO as a pass in `Spatial`'s compiler, so that `Spatial` can automatically use PrBO for the optimization during compilation. This enables `Spatial` to seamlessly call PrBO during the compilation of any new application to guide the search towards the best design on an application-specific basis.

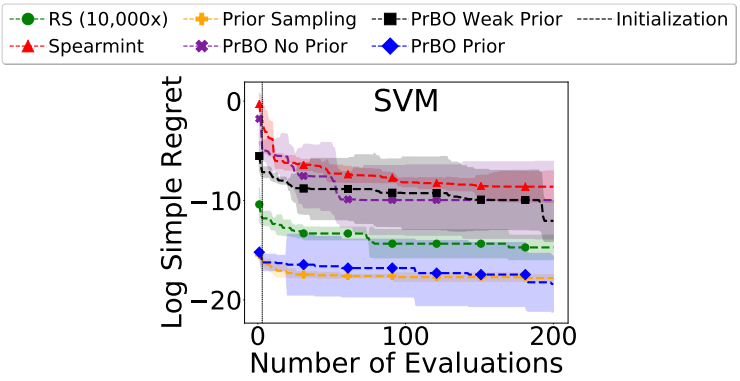


Figure 9: Log regret comparison of PrBO with and without priors, $10,000\times$ RS, and Spearmint (mean +/- one std on 5 repetitions). We run the benchmark for 200 iterations.

Table 2: Search space, priors, and expert configuration for the MD Grid application. The default value for each parameter is shown in bold.

Parameter name	**Type**	**Values**	**Expert**	**Prior**
loop_grid0_z	Ordinal	[**1**, 2, ..., 15, 16]	1	[0.2, 0.1, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05]
loop_q	Ordinal	[**1**, 2, ..., 31, 32]	8	[0.08, 0.08, 0.02, 0.1, 0.02, 0.02, 0.02, 0.1, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.1, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02]
par_load	Ordinal	[**1**, 2, 4]	1	[0.45, 0.1, 0.45]
loop_p	Ordinal	[**1**, 2, ..., 31, 32]	2	[0.1, 0.1, 0.1, 0.1, 0.05, 0.03, 0.02]
loop_grid0_x	Ordinal	[**1**, 2, ..., 15, 16]	1	[0.2, 0.1, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05]
loop_grid1_z	Ordinal	[**1**, 2, ..., 15, 16]	1	[0.2, 0.2, 0.1, 0.1, 0.07, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03]
loop_grid0_y	Ordinal	[**1**, 2, ..., 15, 16]	1	[0.2, 0.1, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05]
ATOM1LOOP	Categorical	[false, **true**]	true	[0.1, 0.9]
ATOM2LOOP	Categorical	[false, **true**]	true	[0.1, 0.9]
PLOOP	Categorical	[false, **true**]	true	[0.1, 0.9]

In our experiments, we introduce for the first time the automatic optimization of three `Spatial` real-world applications, namely, 7D shallow and deep CNNs, and a 10D molecular dynamics grid application. Previous work by Nardi et al. (2019) had applied automatic optimization of `Spatial` parameters on a set of benchmarks but in our work we focus on real-world applications raising the bar of state-of-the-art automated hardware design optimization. PrBO is used to optimize the parameters to find a design that leads to the fastest runtime. The search space for these three applications is based on ordinal and categorical parameters; to handle these discrete parameters in the best way we implement and use a Random Forest surrogate instead of a Gaussian Process one as explained in Appendix D. These parameters are application specific and control how much of the FPGAs' resources we want to use to parallelize each step of the application's computation. The goal here is to find which steps are more important to parallelize in the final design, in order to achieve the fastest

Table 3: Search space, priors, and expert configuration for the Shallow CNN application. The default value for each parameter is shown in bold.

Parameter name	Type	Values	Expert	Prior
LP	Ordinal	[**1**, 4, 8, 16, 32]	16	[0.4, 0.065, 0.07, 0.065, 0.4]
P1	Ordinal	[**1**, 2, 3, 4]	1	[0.1, 0.3, 0.3, 0.3]
SP	Ordinal	[**1**, 4, 8, 16, 32]	16	[0.4, 0.065, 0.07, 0.065, 0.4]
P2	Ordinal	[**1**, 2, 3, 4]	4	[0.1, 0.3, 0.3, 0.3]
P3	Ordinal	[**1**, 2, ..., 31, 32]	1	[0.1, 0.1, 0.033, 0.1, 0.021, 0.021, 0.021, 0.1, 0.021]
P4	Ordinal	[**1**, 2, ..., 47, 48]	4	[0.08, 0.0809, 0.0137, 0.1, 0.0137, 0.0137, 0.0137, 0.1, 0.0137, 0.0137, 0.0137, 0.05, 0.0137]
x276	Categorical	[false, **true**]	true	[0.1, 0.9]

Table 4: Search space, priors, and expert configuration for the Deep CNN application. The default value for each parameter is shown in bold.

Parameter name	Type	Values	Expert	Prior
LP	Ordinal	[**1**, 4, 8, 16, 32]	8	[0.4, 0.065, 0.07, 0.065, 0.4]
P1	Ordinal	[**1**, 2, 3, 4]	1	[0.4, 0.3, 0.2, 0.1]
SP	Ordinal	[**1**, 4, 8, 16, 32]	8	[0.4, 0.065, 0.07, 0.065, 0.4]
P2	Ordinal	[**1**, 2, 3, 4]	2	[0.4, 0.3, 0.2, 0.1]
P3	Ordinal	[**1**, 2, ..., 31, 32]	1	[0.04, 0.01, 0.01, 0.1, 0.01, 0.01, 0.01, 0.1, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.2, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.1, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.2]
P4	Ordinal	[**1**, 2, ..., 47, 48]	4	[0.05, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.13, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.2, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.11, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.2, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.1]
x276	Categorical	[false, **true**]	true	[0.1, 0.9]

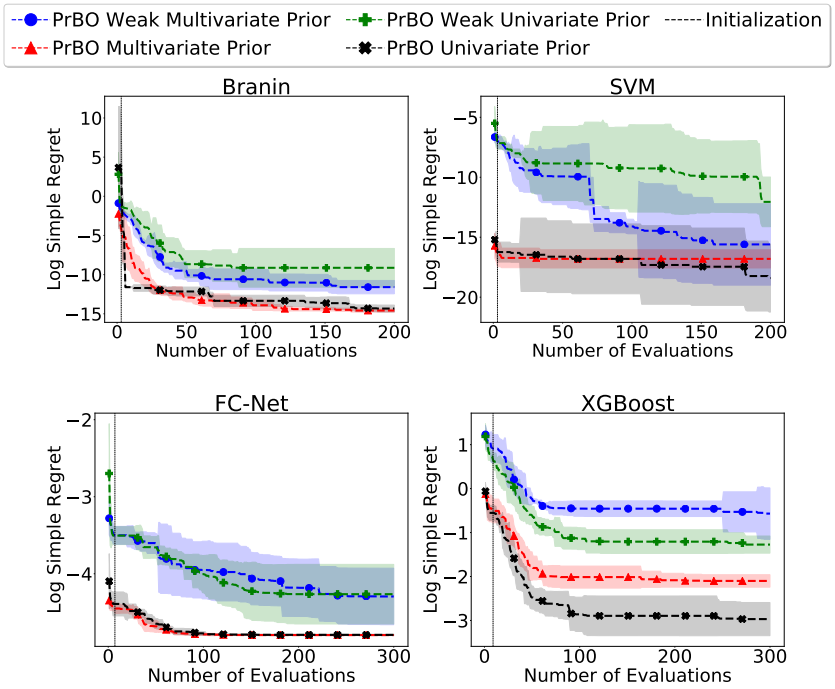


Figure 10: Log regret comparison of PrBO with multivariate and univariate KDE priors. The line and shaded regions show the mean and standard deviation of the log simple regret after 5 runs. All methods were initialized with $D + 1$ random samples, where D is the number of input dimensions, indicated by the vertical dashed line. We run the benchmarks for $100D$ iterations, capped at 300.

runtime. Some parameters also control whether we want to enable pipeline scheduling or not, which consumes resources but accelerates runtime. We refer to Koeplinger et al. (2018) and Nardi et al. (2019) for more details on `Spatial`'s parameters.

The three `Spatial` benchmarks also have feasibility constraints in the search space, meaning that some parameter configurations are infeasible. A configuration is considered infeasible if the final design requires more logic resources than what the FPGA provides, i.e., it is not possible to perform FPGA synthesis because the design does not fit in the FPGA. To handle these constraints, we use our cBO implementation (Appendix D). Our goal is thus to find the design with the fastest runtime under the constraint that the design fits the FPGA resource budget.

The priors for these `Spatial` applications take the form of a list of probabilities, containing the probability of each ordinal or categorical value being good. Each benchmark also has a default configuration, which ensures all methods start with at least one feasible configuration. The priors and the default configuration for these benchmarks were provided once by an unbiased `Spatial` developer, who is not an author of this paper, and kept unchanged during the entire project. The search space, priors, and the expert configuration used in our experiments for each application are presented in Tables 2, 3, and 4.

G Multivariate Prior Comparison

Figure 10 shows a log regret comparison of PrBO with univariate and multivariate KDE priors. We show results for univariate and multivariate versions of our weak and strong KDE priors. We use the best $10D$ points out of $1{,}000D$ and $10{,}000{,}000D$ randomly sampled points to create our weak and strong priors, respectively. We use the same points to create the univariate and multivariate priors. We recall that the goal of these synthetic priors is to have an unbiased prior for our experiments, whereas manual priors would be biased by our own expertise of these benchmarks. In practice, users will manually define these priors without needing additional experiments.

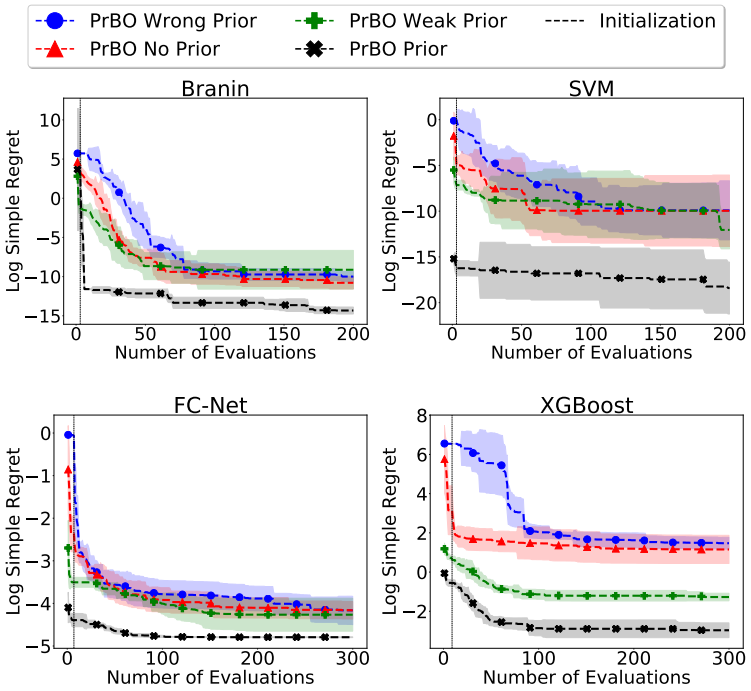


Figure 11: Log regret comparison of PrBO with varying prior quality. The line and shaded regions show the mean and standard deviation of the log simple regret after 5 runs. All methods were initialized with $D + 1$ random samples, where D is the number of input dimensions, indicated by the vertical dashed line. We run the benchmarks for $100D$ iterations, capped at 300.

We note that in all cases PrBO achieves similar performance with univariate and multivariate priors. For the Branin and SVM benchmarks, the weak multivariate prior leads to slightly better results than the weak univariate prior. However, we note that the difference is small, in the order of 10^{-4} and 10^{-6}, respectively.

Surprisingly, for the XGBoost benchmark, the univariate version for both the weak and strong priors lead to better results than their respective multivariate counterparts, though, once again, the difference in performance is small, around 0.2 and 0.03 for the weak and strong prior, respectively, whereas the XGBoost benchmark can reach values as high as $f(\boldsymbol{x}) = 600$. Our hypothesis is that this difference comes from the bandwidth estimator ($100n^{-\frac{1}{D}}$), which leads to larger bandwidths, consequently, smoother priors, when a multivariate prior is constructed.

H Misleading Prior Comparison

Figure 11 shows the effect of injecting a misleading prior in PrBO. We compare PrBO with a misleading prior, no prior, a weak prior, and a strong prior. To create the misleading prior, we use a univariate KDE prior built with the worst $10D$ out of $10{,}000{,}000D$ random samples. For all benchmarks, we note that the misleading prior slows down convergence, as expected, since it pushes the optimization away from the optima in the initial phase. However, PrBO is still able to forget the misleading prior and achieve similar regret to PrBO without a prior.

I Comparison to Other Baselines

We compare PrBO to SMAC (Hutter et al., 2011) and TPE (Bergstra et al., 2011) on our four synthetic benchmarks. We use Hyperopt's implementation[4] of TPE and the SMAC3 Python implementation

[4]https://github.com/hyperopt/hyperopt

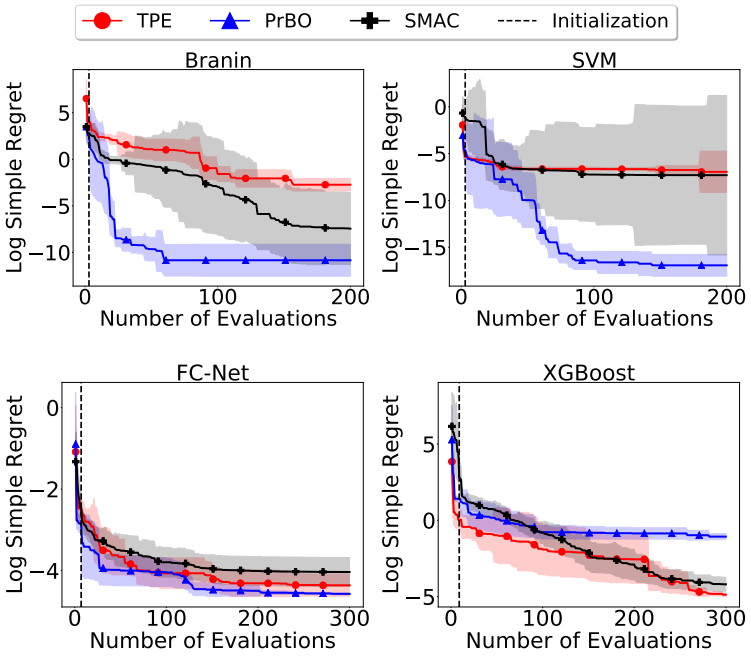


Figure 12: Log regret comparison of PrBO, SMAC, and TPE. The line and shaded regions show the mean and standard deviation of the log simple regret after 5 runs. PrBO was initialized with $D + 1$ random samples, where D is the number of input dimensions, indicated by the vertical dashed line. We run the benchmarks for $100D$ iterations, capped at 300.

of SMAC[5]. Hyperopt defines priors as one of a list of supported distributions, including Uniform, Normal, and Lognormal distributions. Since it is not possible to input the KDE priors introduced in Section 4.1 into the TPE algorithm, we instead use manually defined priors in the format supported by the HyperOpt implementation. We note that this is straightforward in PrBO, as PrBO supports any form of probability distribution as a prior. We are then able to perform a fair comparison between the two approaches that use the same exact prior. Since SMAC does not support probability distribution priors, we do not inject any priors in SMAC.

We define the prior for each input parameter as a Gaussian distribution with mean at the optimum and with standard deviation equal to half of the parameters range. For the Branin prior, we arbitrarily choose one of the optima, i.e., the $(\pi, 2.275)$ optimum. For the Profet benchmarks, we use the minimum out of $10{,}000{,}000D$ random samples as an approximation of the optimum. We note that using Hyperopt's Gaussian priors leads to an unbounded search space, which sometimes leads TPE to suggest parameter configurations outside the allowed parameter range. To prevent these values from being evaluated, we convert values outside the parameter range to be equal to the upper or lower range limit, depending on which limit was exceeded.

Figure 12 shows a log regret comparison between PrBO, SMAC, and TPE on our four synthetic benchmarks. PrBO outperforms SMAC on three out of four benchmarks, and shows consistent speedups in the early phases of the optimization proces. PrBO also outperforms TPE in three out of four benchmarks, namely, Branin, SVM, and FCNet. We note, however, that the good performance of TPE on XGBoost may be an artifact of the approach of clipping values to its maximal or minimal values as mentioned above. In fact, the clipping nudges TPE towards promising configurations in this case, since XGBoost has low function value near the edges of the search space. Overall, the better performance of PrBO is expected, since PrBO is able to combine prior knowledge with more sample-efficient surrogates, which leads to better performance.

[5]https://github.com/automl/SMAC3

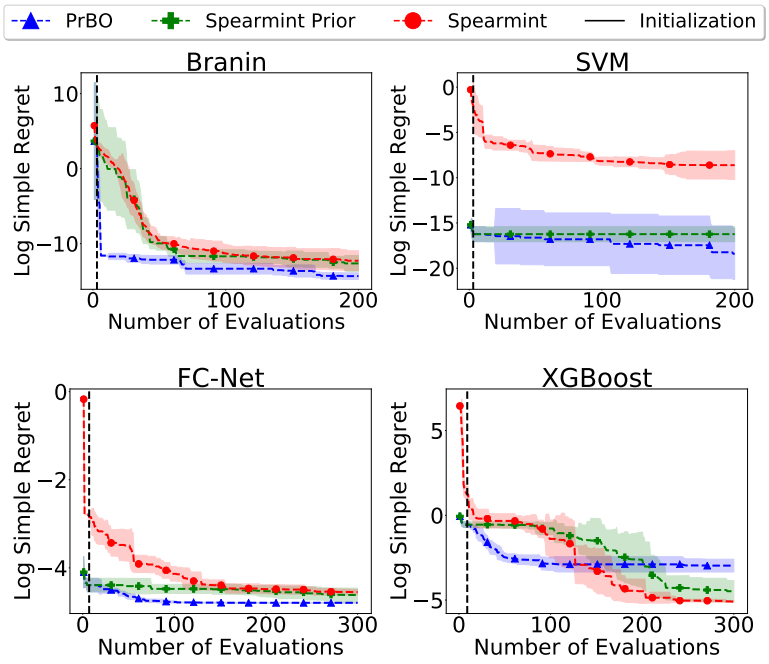


Figure 13: Log regret comparison of PrBO, Spearmint with prior initialization, and Spearmint with default initialization. The line and shaded regions show the mean and standard deviation of the log simple regret after 5 runs. PrBO and Spearmint Prior were initialized with $D + 1$ random samples from the prior, where D is the number of input dimensions, indicated by the vertical dashed line. We run the benchmarks for $100D$ iterations, capped at 300.

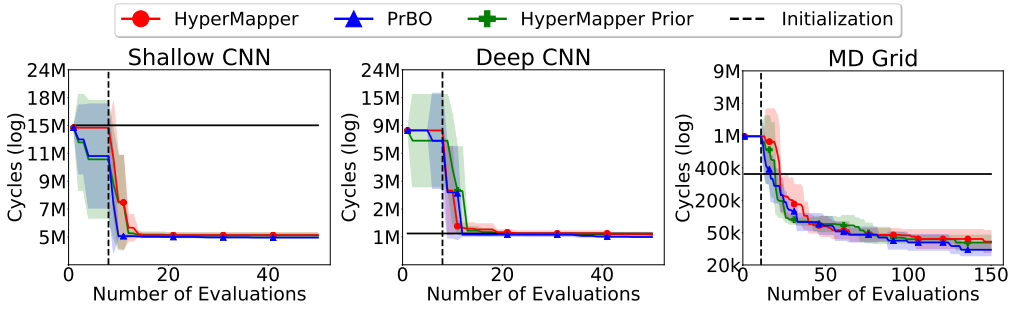


Figure 14: Log regret comparison of PrBO, HyperMapper with prior initialization, and HyperMapper with default initialization. The line and shaded regions show the mean and standard deviation of the log simple regret after 5 runs. PrBO and HyperMapper Prior were initialized with $D + 1$ random samples from the prior, where D is the number of input dimensions, indicated by the vertical dashed line.

J Prior Baselines Comparison

In this section, we compare PrBO with Spearmint and HyperMapper using the same prior initialization on all of our benchmarks (Figs. 13 and 14). Both PrBO and the baselines were initialized with the same $D + 1$ samples from the prior, indicated by the vertical dashed line. We run each optimizer for 5 times and report mean and standard deviation of the 5 runs. For comparison, we also show the baselines with the default initialization.

Figure 13 shows the comparison between PrBO and Spearmint Prior. In most benchmarks, the prior initialization does not lead to improved final performance. In particular, for XGBoost, the prior leads

to improvement in early iterations, but to worse final performance. We note that for SVM, the prior leads to better performance, however, we note that the improved performance is given solely from sampling form the prior. There is no improvement for Spearmint Prior after initialization. In contrast, in all cases, PrBO is able to leverage the prior both during initialization and its Bayesian Optimization phase, leading to improved performance. PrBO still outperforms Spearmint Prior in the same 3 out of 4 benchmarks.

Figure 14 shows similar results for our Spatial benchmarks. Once again, the prior does not lead HyperMapper to improved final performance. For the Shallow CNN benchmark, the prior leads HyperMapper to improved performance in early iterations, compared to HyperMapper with default initialization, but HyperMapper Prior is still outperformed by PrBO. Additionaly, the prior leads to degraded performance in the Deep CNN benchmark. These results confirm that PrBO is able to leverage the prior in its pseudo-posterior during optimization, leading to improved performance in almost all benchmarks compared to state-of-the-art BO baselines.

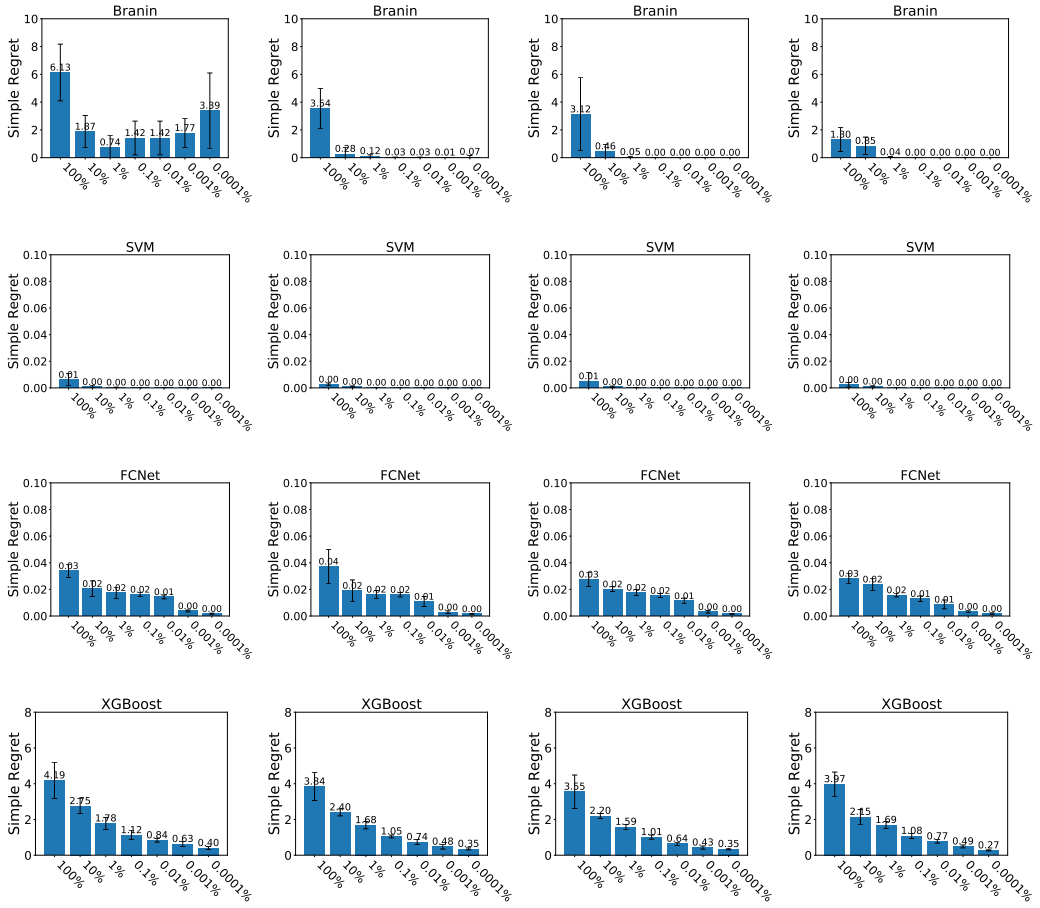
K PRIOR BANDWIDTH SELECTION

We show the effect of different bandwidth sizes on the univariate KDE prior. For that, we compare the performance of sampling from the prior and PrBO with different bandwidth sizes. We use scipy's Gaussian KDE implementation and modify its bandwidth size with four variations of Scott's Rule $an^{-\frac{1}{D+b}}$. We experiment with $a = 1$, $b = 4$ (scipy's default); $a = 1$, $b = 0$; $a = 10$, $b = 0$; and $a = 100$, $b = 0$. Note that larger values for a and smaller values for b lead to smaller bandwidths. For each bandwidth size, we show results for an array of varying quality priors. We select a constant $10D$ points in each prior and vary the size of the uniform random sample dataset. We follow the following rule: we use the best performing $10D$ samples to create the prior from a random sample dataset size of $10D\frac{100}{x}$; we refer to this prior as $x\%$. We experiment with dataset sizes varying from $10D$ to $10^7 D$.

Figure 15 shows the performance of purely sampling from the prior. We note that, in most cases, using a larger dataset leads to better results. This is expected, sampling more points means we find more points near the optima and, therefore, the prior will be built with points closer to the optima. Likewise, we note that smaller bandwidths often lead to better results, especially as more points are sampled. This is also expected, since a smaller bandwidth means the prior distribution will be more peaked around the optima. However, there are a couple of exceptions to these trends. First, for the Branin, sampling more points does not lead to a better prior when we use $a = 1, b = 4$, this is likely because the multiple minima of the Branin and the bigger bandwidth lead the prior to be oversmoothed, missing the peaks near the optima. Second, smaller bandwidths do not always lead to better performance for smaller random sample datasets. This happens because we find points farther from the optima in these datasets and end up computing priors peaked at points that are farther from the optima, i.e., our priors become misleading. The effects of these misleading priors can be especially noticed for the 100% random samples dataset. Based on these results, we set our KDE priors to $100n^{-\frac{1}{D}}$, where D is the number of input dimensions.

Figure 16 shows the performance of PrBO for different priors. The same observations from Figure 15 hold here. Namely, sampling more points and using smaller bandwidths lead to better performance. Also, the 100% dataset once again leads to inconsistent results, since it is a misleading prior for PrBO. Based on these results, we use the smallest bandwidth and largest dataset in our experiments, i.e. $a = 100, b = 0$, and 0.0001%. Intuitively, this is a reasonable choice, since these priors will be our closest approximation to an ideal prior that is centered exactly at the optima, where sampling from the prior always leads to the optimum. Our results in Figures 15 and 16 shows that this combination leads to the best results in all benchmarks, as expected.

Figure 17 shows a performance comparison between PrBO and sampling from the prior. For these results, we use $a = 100, b = 0$ and compare the regret of PrBO and sampling from the prior for different dataset sizes. PrBO performs better for nearly all dataset sizes and benchmarks. This is expected as PrBO complements the prior with its probabilistic model, learning which regions within the prior are better to explore and also recovering from misleading priors. There are two exceptions on the SVM benchmark, where sampling from the prior performs slightly better for 0.01% and



(a) $a = 1, b = 4$ (b) $a = 1, b = 0$ (c) $a = 10, b = 0$ (d) $a = 100, b = 0$

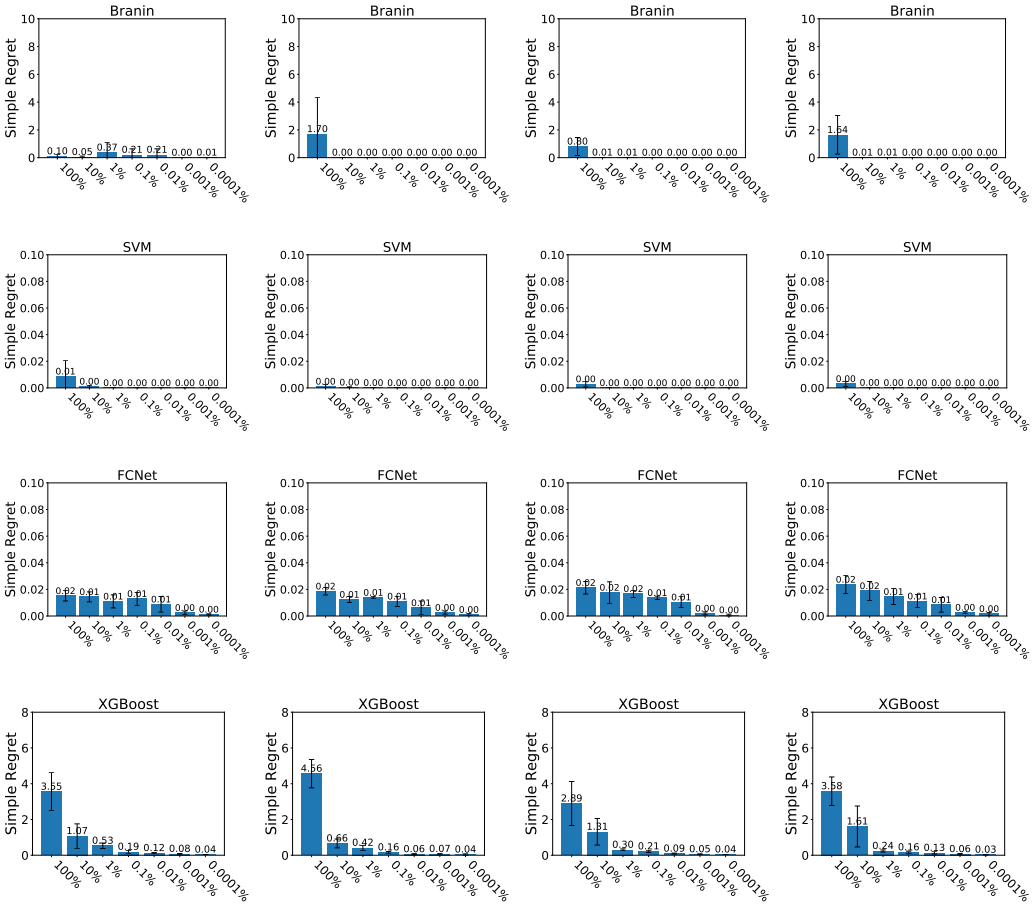
Figure 15: Simple regret of sampling from the prior with different priors for our synthetic benchmarks. We provide 5 repetitions for each experiment and mean +/- one std error bars. A more informative prior gives better results in all benchmarks.

0.0001% datasets. We note, however, that the difference in performance is extremely small, in the order of 10^{-5} and 10^{-7}, respectively.

L γ-SENSITIVITY STUDY

We show the effect of the γ hyperparameter introduced in Section 3.2 for the quantile identifying the points considered to be good. To show this, we compare the performance of PrBO with a weak KDE prior and different γ values. We use our weak prior as it leads to greater variation in performance, which helps to visualize better the impact of the γ hyperparameter. For all experiments, we initialize PrBO with $D + 1$ random samples and then run PrBO until it reaches $10D$ function evaluations. For each γ value, we run PrBO five times and report mean and standard deviation.

Figure 18 shows the results of our comparison. We first note that values near the lower and higher extremes lead to degraded performance, this is expected, since these values will lead to an excess of either exploitation or exploration. Further, we note that PrBO achieves similar performance for all values of γ, however, $\gamma = 0.03$ and $\gamma = 0.05$ consistently lead to better performance, with $\gamma = 0.05$ usually leading to lower deviation.



(a) $a = 1, b = 4$ (b) $a = 1, b = 0$ (c) $a = 10, b = 0$ (d) $a = 100, b = 0$

Figure 16: Simple regret of PrBO with different priors for our synthetic benchmarks. We provide 5 repetitions for each experiment and mean +/- one std error bars. A more informative prior gives better results in all benchmarks.

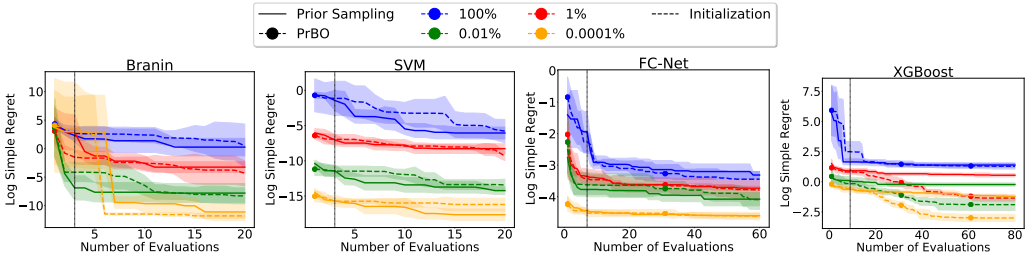


Figure 17: Log simple regret comparison between PrBO and sampling from the prior. The shaded lines are mean +/- one std error. PrBO was initialized with $D + 1$ random samples, indicated by the vertical dashed line.

M β-SENSITIVITY STUDY

We show the effect of the β hyperparameter introduced in Section 3.3 for controlling the influence of the prior over time. To show the effects of β, we compare the performance of PrBO with a weak KDE prior and different β values on our four synthetic benchmarks. We use our weak prior as it leads to

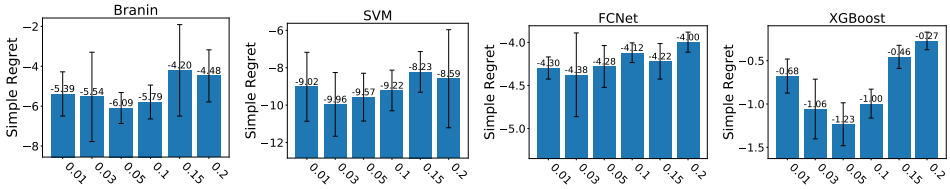


Figure 18: Comparison of PrBO with a weak KDE prior and different values for the γ hyperparameter on our four synthetic benchmarks. We run PrBO with a budget of $10D$ function evaluations, including $D + 1$ randomly sampled DoE configurations.

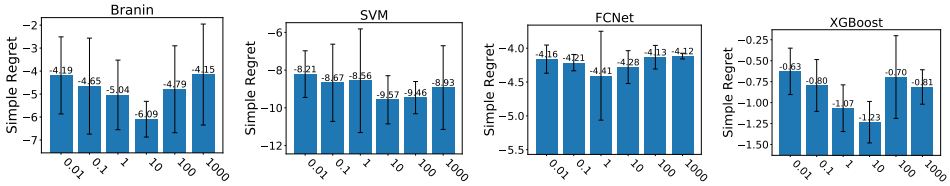


Figure 19: Comparison of PrBO with a weak KDE prior and different values for the β hyperparameter on our four synthetic benchmarks. We run PrBO with a budget of $10D$ function evaluations, including $D + 1$ randomly sampled DoE configurations.

greater variation in performance, which helps to visualize better the impact of the β hyperparameter. For all experiments, we initialize PrBO with $D + 1$ random samples and then run PrBO until it reaches $10D$ function evaluations. For each β value, we run PrBO five times and report mean and standard deviation.

Figure 19 shows the results of our comparison. We note that values of β that are too low (near 0.01) or too high (near 1000) lead to lower performance. This shows that putting too much emphasis on the model or the prior will lead to degraded performance, as expected. Further, we note that $\beta = 10$ lead to the best performance in three out of our four benchmarks. This result is reasonable, as $\beta = 10$ means PrBO will put more emphasis on the prior in early iterations, when the predictive model is still not accurate, and slowly shift towards putting more emphasis on the model as the model sees more data and becomes more accurate.