Prior-guided Bayesian Optimization

Artur Souza
Universidade Federal de Minas Gerais
arturluis@dcc.ufmg.br

Luigi Nardi
Lund University
Stanford University
luigi.nardi@cs.lth.se

Leonardo B. Oliveira
Universidade Federal de Minas Gerais
leob@dcc.ufmg.br

Kunle Olukotun
Stanford University
kunle@stanford.edu

Marius Lindauer
Leibniz University Hannover
lindauer@tnt.uni-hannover.de

Frank Hutter
University of Freiburg
Bosch Center for Artificial Intelligence
fh@cs.uni-freiburg.de

Abstract

While Bayesian Optimization (BO) is a very popular method for optimizing expensive black-box functions, it fails to leverage the knowledge of domain experts. This causes BO to waste function evaluations on bad design choices (e.g., machine learning hyperparameters) that the expert already knows to work poorly. To address this issue, we introduce Prior-guided Bayesian Optimization (PrBO). PrBO allows users to transfer their knowledge into the optimization process in the form of priors about which parts of the input space will yield the best performance, rather than BO's standard priors over functions (which are much less intuitive for users). PrBO then combines these priors with BO's standard probabilistic model to form a pseudo-posterior used to select which points to evaluate next. We show that PrBO is around $12\times$ faster than state-of-the-art methods without user priors and $10{,}000\times$ faster than random search on a common suite of benchmarks. PrBO also converges faster even if the user priors are not entirely accurate and robustly recovers from misleading priors.

1 Introduction

Bayesian Optimization (BO) is a data-efficient method for the joint optimization of design choices that gained great popularity in recent years. It is impacting a wide range of areas, including hyperparameter optimization (Snoek et al., 2012; Falkner et al., 2018), AutoML (Feurer et al., 2015a; Hutter et al., 2018), robotics (Calandra et al., 2016), computer vision (Nardi et al., 2017; Bodin et al., 2016), Computer Go (Chen et al., 2018), hardware design (Koeplinger et al., 2018; Nardi et al., 2019), and many others. It promises greater automation so as to increase both product quality and human productivity. As a result, BO is also established in many large tech companies, e.g., with Google Vizier (Golovin et al., 2017) and Facebook BoTorch (Balandat et al., 2019).

Nevertheless domain experts often have substantial prior knowledge that standard BO cannot incorporate. Users can incorporate prior knowledge by narrowing the search space; however, this type of hard prior can lead to poor performance by missing important regions. BO also supports a prior over functions $p(f)$, e.g., via a kernel function. However, this is not the prior experts have: users often know which ranges of hyperparameters tend to work best, and are able to specify a probability

4th Workshop on Meta-Learning at NeurIPS 2020, Vancouver, Canada.

distribution $p_{best}(x)$ to quantify these priors. E.g., many users of the Adam optimizer (Kingma & Ba, 2015) know that its best learning rate is often in the vicinity of 1e-3. Similarly, Navruzyan et al. (2019) derived neural network hyperparameter priors for image datasets based on their experience with ve datasets. In these cases, users know potentially good values for a new application, but cannot be certain about them.

As a result, many competent users instead revert to manual search, which can fully incorporate their prior knowledge. A recent survey showed that most NeurIPS 2019 and ICLR 2020 papers that reported having tuned hyperparameters used manual search, with only a very small fraction using BO (Bouthillier & Varoquaux, 2020). In order for BO to be adopted widely, and help facilitate faster progress in the ML community by tuning hyperparameters faster and better, it is therefore crucial to devise a method that allows experts to fully transfer their knowledge into the optimization. In this paper, we introduce Prior-guided Bayesian Optimization (PrBO), a novel BO variant that allows users to transfer their knowledge into BO in the form of priors. PrBO then combines this prior knowledge with its learning model in order to learn better and faster where to nd promising hyperparameter con gurations. Our technical contributions with PrBO are:

1. PrBO bridges the TPE methodology and standard BO probabilistic models, such as GPs, RFs or Bayesian NNs, instead of Tree Parzen Estimators only.
2. PrBO is exible w.r.t. how the prior is de ned, allowing previously hard-to-inject (e.g. exponential) priors.
3. PrBO gives more importance to the model as iterations progress, gradually forgetting the prior and ensuring robustness against misleading priors.

We demonstrate the effectiveness of PrBO on a common suite of benchmarks, showing that accurate prior knowledge helps PrBO to achieve similar performance to current state-of-the-art on average 12 faster. PrBO also achieves better nal performance in all but one of the benchmarks tested.

2 Background: Tree-structured Parzen Estimator

Our work is partially inspired by the TPE methodology of Bergstra et al. (2011). Whereas the standard probabilistic model in BO directly models $p(y|x)$, the Tree-structured Parzen Estimator (TPE) approach of Bergstra et al. models $p(x|y)$ and $p(y)$ instead[1]. This is done by constructing two parametric densities $g(x)$ and $l(x)$, which are computed using the observations with function value above and below a given threshold, respectively. The separating threshold y^* is de ned as a quantile of the observed function values. TPE then de nes $p(x|y)$ as:

$$p(x|y) = l(x)\mathbb{I}(y < y^*) + g(x)(1 - \mathbb{I}(y < y^*)); \tag{1}$$

where $\mathbb{I}(y < y^*)$ is 1 when $y < y^*$ and 0 otherwise. The parametrization of the generative model $p(x, y) = p(x|y)p(y)$ facilitates the computation of EI as it leads to $EI_{y^*}(x) \propto l(x)/g(x)$ and, thus, $\arg\max_{x \in X} EI_{y^*}(x) = \arg\max_{x \in X} l(x)/g(x)$.

3 Bayesian Optimization with Priors

We propose a BO approach dubbed PrBO that allows eld experts to transfer prior knowledge into the optimization in the form of priors. PrBO combines this user-de ned prior with a probabilistic model that captures the likelihood of the observed data $(x_i, y_i)_{i=1}^n$. PrBO is independent from the probabilistic model being used, i.e., it can be freely combined with, e.g., GPs, RFs or BNNs.

3.1 Priors

PrBO allows users to transfer prior knowledge into BO. This is done via a prior distribution that informs where in the input space X we expect to nd good $f(x)$ values. A point is considered "good" if it leads to low function values. We denote the prior distribution $P_g(x)$, where g denotes that this is

[1] Note that, technically, the model does not parameterize $p(y)$, since it is computed based on the observed data points, which are heavily biased towards low values due to the optimization process. Instead, it parameterizes a dynamically changing $p(y_i)_{i=1}^t$, which helps to constantly challenge the model to yield better observations.

a prior on good points and $x \in X$ is a given point. Similarly, we define a prior on where in the input space we expect to have "bad" points. Although we could have a user-defined probability distribution $P_b(x)$, we aimed to keep the load on users low and thus, for simplicity, compute $P_b(x) = 1 - P_g(x)$ ($P_g(x)$ is normalized to $[0, 1]$ by min-max scaling before computing $P_b(x)$).[2]

In practice, x contains several dimensions but it is difficult for experts to provide a multivariate distribution $P_g(x)$. Users can easily specify, e.g., draw, a univariate or bivariate probability distribution for continuous dimensions or provide a list of probabilities for discrete dimensions. In PrBO, users are free to define a complex multi-variate distribution, but we expect the standard use case to be that users only want to specify univariate distributions, implicitly assuming a prior that factors as $P_g(x) = \prod_{i=1}^{D} P_g(x_i)$, where D is the number of dimensions in X, x_i is the i-th input dimension of X. We show examples of continuous and discrete priors in Appendices A and E, respectively. We use factorized priors in our experiments to mimic what we expect most users will provide. In Appendix F, we show that these factorized priors lead to similar performance compared to multivariate priors.

3.2 Model

Whereas the standard probabilistic model in BO, e.g., a GP, quantifies $p(y|x)$ directly, that model is hard to combine with the user-defined prior $P_g(x)$. We therefore introduce a method to translate the standard probabilistic model $p(y|x)$ into a model that is easier to combine with this prior. Similar to the TPE work (see Sec. 2), our model combines $p(x|y)$ and $p(y)$ instead of directly modeling $p(y|x)$.

The computation we perform for this translation is to quantify the probability that a given input x is "good" under our standard probabilistic model $p(y|x)$. As in TPE, we define settings as "good" if their observed y-value is below a certain quantile γ of the observed function values (so that $p(y < f_\gamma) = \gamma$). We in addition exploit the fact that our standard probabilistic model $p(y|x)$ has a Gaussian form, and under this Gaussian prediction we can compute the probability $M_g(x)$ of the function value lying below a certain quantile using the standard closed-form formula for Probability of Improvement (PI, Kushner (1964)):

$$M_g(x) = p(f(x) < f_\gamma \mid x, (x_i, y_i)_{i=1}^t) = \Phi\left(\frac{f_\gamma - \mu_x}{\sigma_x}\right); \tag{2}$$

where μ_x and σ_x are the mean and standard deviation of the probabilistic model at x, and Φ is the standard normal CDF, see Figure 1. Note that there are two probabilistic models here:

> The standard probabilistic model of BO, with a prior over functions $p(f)$, updated by data $(x_i, y_i)_{i=1}^t$ to yield a posterior over functions $p(f \mid (x_i, y_i)_{i=1}^t)$, allowing us to quantify the probability $M_g(x) = p(f(x) < f_\gamma \mid x, (x_i, y_i)_{i=1}^t)$ in Equation 2
>
> The TPE-like generative model that combines $p(x|y)$ and $p(x|y)$ instead of directly modelling $p(y|x)$.

Equation 2 bridges these two models by using the probability of improvement from BO's standard model as the probability $M_g(x)$ in TPE's model. Ultimately, this is a heuristic since there is no formal connection between the two probabilistic models. However, we believe that the use of BO's familiar, theoretically sound framework of probabilistic modelling $p(y|x)$, followed by the computation of the familiar PI formula is a very intuitive choice for obtaining the probability of an input achieving at least a given performance threshold – exactly the term we need for TPE's $M_g(x)$. Similarly, we also define a probability $M_b(x)$ of x being bad as $M_b(x) = 1 - M_g(x)$.

3.3 Pseudo-posterior

PrBO combines the prior in Section 3.1 and the model in Eq. 2 into a pseudo-posterior. It represents the updated beliefs on where we can find good points, based on the prior and data that has been observed. The pseudo-posterior is computed as the product of the prior and the model:

$$g(x) \propto P_g(x) M_g(x)^{\frac{t}{\beta}}; \tag{3}$$

[2] We note that for continuous spaces, this $P_b(x)$ is not a probability distribution, and therefore only a pseudo-prior, as it does not integrate to 1. For discrete spaces, we normalize $P_b(x)$ so that it sums to 1 and therefore is a proper probability distribution and prior.

Algorithm 1 PrBO Algorithm. D keeps track of all function evaluations so far $(x_i, y_i)_{i=1}^{t}$.

Input: Input space X, user-defined prior distributions $P_g(x)$ and $P_b(x)$, quantile γ and BO budget B.
Output: Optimized point x_{inc}.
$D \leftarrow$ Initialize (X)
for $t = 1$ to B do
  $M_g(x) \leftarrow$ fit_model_good(D)
  $M_b(x) \leftarrow$ fit_model_bad(D)
  $g(x) \leftarrow P_g(x) \, M_g(x)^{\frac{t}{\beta}}$
  $b(x) \leftarrow P_b(x) \, M_b(x)^{\frac{t}{\beta}}$
  $x_t \in \arg\max_{x \in X} EI_f(x)$
  $y_t \leftarrow f(x_t)$
  $D = D \cup (x_t, y_t)$
end for
$x_{inc} \leftarrow$ ComputeBest(D)
return x_{inc}

Figure 1: Our model is composed by a probabilistic model and the probability of improving over the threshold γ, i.e., right tail of the Gaussian. The black curve is the probabilistic model's mean and the shaded area is the model's variance.

where t is the current iteration, β is an optimization hyperparameter, $M_g(x)$ is defined in Eq.(2), and $P_g(x)$ is the prior defined in Sec 3.1, rescaled to [0, 1]. We note that this pseudo-posterior is not normalized, but this suffices for PrBO to determine the next x_t as the normalization constant cancels out (c.f. Section 3.4). Since $g(x)$ is not normalized and we add the $\frac{t}{\beta}$ exponent to Eq. 3, we refer to $g(x)$ as a pseudo-posterior, to emphasize that it is not a standard posterior probability distribution.

The $\frac{t}{\beta}$ fraction in Eq.(3) controls how much weight is given to the model. As the optimization progresses, more weight is given to the model over the prior. Intuitively, we put more emphasis on the model as it observes more data and becomes more accurate. We do this under the assumption that the model will eventually be better than the user at predicting where to find good points. This also allows to recover from misleading priors as we show in Appendix A; similar to, and inspired by Bayesian models, the data ultimately washes out the prior. The β hyperparameter defines the balance between prior and model, with higher β values giving more importance to the prior and requiring more data to overrule it.

We note that computing Equation (3) directly can lead to numerical issues. Namely, the pseudo-posterior can reach extremely low values if the prior and model probabilities are low, especially as $\frac{t}{\beta}$ grows. To prevent this, in practice, PrBO uses the logarithm of the pseudo-posterior instead: $\log(g(x)) \propto \log(P_g(x)) + \frac{t}{\beta} \log(M_g(x))$. Once again, we also define an analogous pseudo-posterior distribution on bad: $b(x)$, and use these quantities to define a density model $p(x|y)$:

$$p(x|y) \propto \begin{cases} g(x) & \text{if } y < \gamma \\ b(x) & \text{if } y \geq \gamma \end{cases}. \tag{4}$$

3.4 Acquisition Function

We adopt the EI formulation used in (Bergstra et al., 2011) by replacing their Adaptive Parzen Estimators with our computation of the pseudo-posterior in Eq. (3). Namely, we compute EI as:

$$EI_f(x) := \int_{-\infty}^{\infty} \max(\gamma - y, 0) p(y|x) dy = \int_{-\infty}^{\gamma} (\gamma - y) \frac{p(x|y)p(y)}{p(x)} dy \propto \left(\gamma + \frac{b(x)}{g(x)}(1 - \gamma)\right)^{-1}. \tag{5}$$

The full derivation of Eq.(5) is shown in Appendix B. Eq.(5) shows that to maximize improvement we would like points x with high probability under $g(x)$ and low probability under $b(x)$, i.e., minimizing the ratio $b(x)/g(x)$. We note that the point that minimizes the ratio for our unnormalized pseudo-posteriors will be the same that minimizes the ratio for the normalized pseudo-posterior and, thus, the computation of the normalized pseudo-posteriors is unnecessary.

The dynamics of PrBO can be understood in terms of the following proposition:

Proposition 1 Given f, $P_g(x)$, $P_b(x)$, $M_g(x)$, $M_b(x)$, $g(x)$, $b(x)$, $p(x|y)$, and γ as above, then

$$\lim_{t \to \infty} \arg\max_{x \in X} EI_f(x) = \lim_{t \to \infty} \arg\max_{x \in X} M_g(x);$$

where $M_g(x)$ and EI_f are as defined in Eqs. 2 and 5, respectively.

In early BO iterations the prior will have a predominant role, but in later BO iterations the model will grow more important, and as Proposition 1 shows, if PrBO is run long enough the prior washes out and PrBO only trusts the probabilistic model informed by the data.

3.5 Putting It All Together

Algorithm 1 shows the PrBO algorithm, based on the components defined in the previous sections. In Line 3, PrBO starts with a design of experiments (DoE) phase, where it randomly samples a number of points from the user-defined prior $P_g(x)$. After initialization, the BO loop starts at Line 4. In each loop iteration, PrBO fits the probabilistic model on the previously evaluated points (lines 5 and 6) and computes the pseudo-posteriors $g(x)$ and $b(x)$ (lines 7 and 8 respectively). The EI acquisition function is computed next, using the pseudo-posteriors, and the point that maximizes EI is selected as the next point to evaluate at line 9. The black-box function evaluation is performed at Line 10. This BO loop is repeated for a pre-defined number of iterations, according to the user-defined Budget B.

4 Experiments

We implement both GPs and RFs as predictive models and use GPs in our experiments unless stated otherwise. We set the model weight $\eta = 10$ and the model quantile to $\gamma = 0.05$, based on our sensitivity studies in Appendices I and J. Before starting the main BO loop in PrBO, we randomly sample $D + 1$ points from the prior. We optimize EI using a multi-start local search, similar to SMAC (Hutter et al., 2011). We start with four synthetic benchmarks: Branin, SVM, FC-NET and XGBoost, which are 2, 2, 6 and 8 dimensional, respectively. The last three are part of the Profet benchmarks (Klein et al., 2019), generated by a generative model built using performance data on OpenML or UCI datasets. See Appendix C for more details on the experimental setup. Due to space constraints, we defer the (qualitatively similar) results for the SVM benchmark to Appendix D. For the same reason, we defer to Appendix E the study of a real-world application, a programming language and compiler named Spatial for the design of application accelerators, i.e., FPGAs. We apply PrBO to three Spatial benchmarks, namely, 7D shallow and deep CNNs, and a 10D molecular dynamics grid application. We optimize design runtime constrained to the design fitting a target FPGA. Compared to the previous state-of-the-art, PrBO converges on average 1.49× faster on two benchmarks and achieves 1.28× better final performance on the third. For context, this is a significant improvement in the FPGA field, where 10% improvement could qualify for acceptance in a top-tier conference.

4.1 Prior Selection

In this section we study the effect of choosing a prior. A suitable property of the prior is that, by selecting a tighter prior around an optimum, we would expect sampling from the prior to have an increased performance. To the limit, if the prior is composed by only one point which is one of the global optima, then the first sample (and all of them) from the prior will hit the optimum. To have a sanity check of this property, we build an artificial prior in a controlled way. We rely on an automated computation of the prior by computing a univariate Kernel Density Estimation (KDE) using a Gaussian kernel on the synthetic benchmarks introduced above. We note 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.

We experiment with an array of varying quality priors. We select a constant 10D points in each prior and vary the size of the random sample dataset so that we can make the priors more sharply peaked around the optima in a controlled environment. We use the best performing $10D$ samples to create

Figure 2: Regret of prior sampling and PrBO with different priors (on 5 reps.).

Figure 3: Log regret comparison of PrBO with and without priors, RS, and Spearmint (mean +/- one std on 5 repetitions). We run the benchmarks for 100D iterations, capped at 300.

the prior from a uniform random sample dataset size of $\frac{100}{x}$; we refer to this prior as x% in Figure 2. As an example the XGBoost benchmark has D = 8, so, 100% means we sample 80 points and use all 80 to create the prior, 10% means we sample 800 points and use the best performing 80 to create the prior, 1% means we sample 8,000 and use the best 80 to create the prior, and so on.

Figure 2 shows the performance of purely sampling from the prior and running PrBO, respectively, after 10D function evaluations with different priors. A bigger random sample dataset and a smaller percentage leads to a tighter prior around the optimum, making the argument for a stronger prior. This is con rmed by Figure 2, where a sharply peaked prior (right side of the gure) leads to a better performance in both scenarios. In addition we observe that in contrast to sampling from the prior, PrBO achieves a smaller regret by being able to evolve from the initial prior and making independent steps towards better values of the objective function. More extensive experiments with a similar trend, including the rest of the benchmarks, are in Appendix H.

4.2 Comparison Against Strong Baselines

Figure 3 compares PrBO to other optimizers using the log simple regret on ve runs (mean and std error reported) on the synthetic benchmarks. We consider two priors in our experiments, a strong prior, computed using a KDE on the best 1D out of 10,000,000D uniform random samples, and a weak prior, computed using a KDE on the best 1D out of 1,000D uniform random samples. We emphasize that we only used these arti cial priors in these experiments to guarantee that our prior is not biased by our own expertise for the benchmarks we used. In practice, the prior is de ned by the user. We refer to Appendices A, E, and G for examples of PrBO with different prior types. We compare the results of PrBO with and without priors (both weak and strong), to 100 random search (RS, i.e., for each BO sample we draw 10,000 uniform random samples), sampling from the strong prior only, and Spearmint (Snoek et al., 2012) which is a well-adopted BO approach using GPs and the EI acquisition function.

PrBO prior beats 10,000 RS and PrBO weak prior on all benchmarks. It also either outperforms or matches the performance of sampling from the prior; this is expected because prior sampling cannot recover from a non-ideal prior. The two methods are identical up to the initialization phase because they both sample from the same prior in that phase.

PrBO Prior is more sample ef cient and nds better results than Spearmint on three out of the four benchmarks. On XGBoost, PrBO leads the performance until 139 BO iterations, where Spearmint catches up and achieves better results in the end. Importantly, in all our experiments, PrBO with a good prior consistently shows tremendous speedups in the early phases of the optimization process, typically only requiring on average 8.25 iterations to reach the performance that Spearmint reaches after 100 iterations (12:12 faster). Thus in comparison to traditional BO approaches, PrBO makes use of the best of both worlds, leveraging prior knowledge and ef cient optimization based on BO.

5 Related Work

TPE by Bergstra et al. (2011) supports limited hand-designed priors in the form of normal or log-normal distributions. We make three technical contributions that make PrBO more exible than TPE. First, we generalize over the TPE approach by allowing more exible priors; second, our approach is

model-agnostic (i.e., PrBO is not limited to the TPE model; we use GPs and RFs in our experiments); and third, PrBO is inspired by Bayesian models that give more importance to the data as iterations progress. We also show that PrBO outperforms HyperOpt's TPE in Appendix G.

In parallel work, Li et al. (2020) also allow users to specify priors via a probability distribution. Their two-level approach first samples a number of configurations by maximizing Thompson samples from a GP posterior and then chooses the configuration that has the highest prior as the next to evaluate. In contrast, our method leverages the information from the prior more directly and ensures that the prior gets washed out as we collect more data, enabling PrBO to overcome misspecified priors.

Similarly, black-box optimization tools, such as SMAC (Hutter et al., 2011) or iRace (López-Ibáñez et al., 2016) also support simple hand-designed priors, e.g. log-transformations. However, these are not properly reflected in the predictive models and both cannot explicitly recover from bad priors.

Our work also relates to other meta-learning for BO approaches (Vanschoren, 2019), where BO is applied to many similar optimization problems in a sequence such that knowledge about the general problem structure can be exploited in future optimization problems. In contrast to these approaches, PrBO is the first method that allows human experts to explicitly specify their priors. Furthermore, PrBO does not depend on any meta-features (Feurer et al., 2015b) and only incorporates a single prior instead of many priors from different experiments (Lindauer & Hutter, 2018).

6 Conclusions and Future Work

We have proposed a novel BO variant, PrBO, that allows users to transfer their expert knowledge into the optimization in the form of priors about which parts of the input space will yield the best performance. These are different than common priors over functions which are much less intuitive for users. BO failed so far to leverage the knowledge of domain experts, not only causing inefficiency but also getting users away from applying BO approaches because they could not exploit their prior knowledge. PrBO addresses this issue and will therefore facilitate the adoption of BO. We showed that PrBO is 12.12x more sample efficient than state-of-the-art methods, and 1,000s faster than random search, on a common suite of benchmarks, and also achieves better final performance in all but one of the benchmarks tested. PrBO also robustly recovers from misleading priors. In future work, we will study how PrBO can be used to leverage other types of prior knowledge from meta-learning, to boost BO's performance even further.

References

Maximilian Balandat, Brian Karrer, Daniel R Jiang, Samuel Daulton, Benjamin Letham, Andrew Gordon Wilson, and Eytan Bakshy. Botorch: Programmable bayesian optimization in pytorch. *arXiv preprint arXiv:1910.06403*, 2019.

James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, pp. 2546–2554, 2011.

Bruno Bodin, Luigi Nardi, M Zeeshan Zia, Harry Wagstaff, Govind Sreekar Shenoy, Murali Emani, John Mawer, Christos Kotselidis, Andy Nisbet, Mikel Lujan, et al. Integrating algorithmic parameters into benchmarking and design space exploration in 3d scene understanding. In *Proceedings of the 2016 International Conference on Parallel Architectures and Compilation*, pp. 57–69, 2016.

Xavier Bouthillier and Gaël Varoquaux. Survey of machine-learning experimental methods at NeurIPS2019 and ICLR2020. Research report, Inria Saclay Ile de France, January 2020. URL https://hal.archives-ouvertes.fr/hal-02447823.

Roberto Calandra, André Seyfarth, Jan Peters, and Marc Peter Deisenroth. Bayesian optimization for learning gaits under uncertainty. *Annals of Mathematics and Artificial Intelligence*, 76(1-2):5–23, 2016.

Yutian Chen, Aja Huang, Ziyu Wang, Ioannis Antonoglou, Julian Schrittwieser, David Silver, and Nando de Freitas. Bayesian optimization in alphago. *CoRR*, abs/1812.06855, 2018.

Laurence Charles Ward Dixon. The global optimization problem: an introduction. *Toward global optimization*, 2:1–15, 1978.

Stefan Falkner, Aaron Klein, and Frank Hutter. BOHB: robust and ef cient hyperparameter optimization at scale. In Proceedings of the 35th International Conference on Machine Learning, pp. 1436–1445, 2018.

Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Ef cient and robust automated machine learning. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (eds.), Advances in Neural Information Processing Systems 28, pp. 2962–2970. Curran Associates, Inc., 2015a.

Matthias Feurer, Jost Tobias Springenberg, and Frank Hutter. Initializing bayesian hyperparameter optimization via meta-learning. In Proceedings of the Twenty-Ninth AAAI Conference on Arti cial Intelligence, pp. 1128–1135, 2015b.

Jacob R Gardner, Matt J Kusner, Zhixiang Eddie Xu, Kilian Q Weinberger, and John P Cunningham. Bayesian optimization with inequality constraints. In Proceedings of the 31st International Conference on Machine Learning, ICML, 2014.

Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and D. Sculley. Google vizier: A service for black-box optimization. In Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2017.

GPy. GPy: A gaussian process framework in python. http://github.com/SheffieldML/GPy, since 2012.

F. Hutter, L. Xu, H. Hoos, and K. Leyton-Brown. Algorithm runtime prediction: Methods & evaluation. Arti cial Intelligence, 206:79–111, 2014.

Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm con guration. In International conference on learning and intelligent optimization, pp. 507–523. Springer, 2011.

Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren (eds.). Automated Machine Learning: Methods, Systems, Challenges. Springer, 2018. In press, available at http://automl.org/book.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun (eds.), 3rd International Conference on Learning Representations, ICLR, 2015.

Aaron Klein, Zhenwen Dai, Frank Hutter, Neil D. Lawrence, and Javier Gonzalez. Meta-surrogate benchmarking for hyperparameter optimization. In Advances in Neural Information Processing Systems NeurIPS, pp. 6267–6277, 2019.

David Koeplinger, Matthew Feldman, Raghu Prabhakar, Yaqi Zhang, Stefan Hadjis, Ruben Fiszel, Tian Zhao, Luigi Nardi, Ardavan Pedram, Christos Kozyrakis, and Kunle Olukotun. Spatial: A Language and Compiler for Application Accelerators. In ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), June 2018.

Harold J Kushner. A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. Journal of Basic Engineering, 86(1):97–106, 1964.

Cheng Li, Sunil Gupta, Santu Rana, Vu Nguyen, Antonio Robles-Kelly, and Svetha Venkatesh. Incorporating expert prior knowledge into experimental design via posterior sampling. arXiv preprint arXiv:2002.11255, 2020.

Marius Lindauer and Frank Hutter. Warmstarting of model-based algorithm con guration. In Proceedings of the Thirty-Second AAAI Conference on Arti cial Intelligence, pp. 1355–1362, 2018.

Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Thomas Stützle, and Mauro Birattari. The irace package: Iterated racing for automatic algorithm con guration. Operations Research Perspectives, 3:43–58, 2016.

Luigi Nardi, Bruno Bodin, Sajad Saeedi, Emanuele Vespa, Andrew J Davison, and Paul HJ Kelly. Algorithmic performance-accuracy trade-off in 3d vision applications using hypermapper. In 2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), pp. 1434–1443. IEEE, 2017.

Luigi Nardi, David Koeplinger, and Kunle Olukotun. Practical design space exploration. In 27th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, MASCOTS 2019, 2019.

Arshak Navruzyan, Frank Sharp, Jeremy Howard, and Antoine Saliou. Optimizing hyperparams for image datasets in fastai. https://platform.ai/blog/page/1/optimizing-hyperparams-for-image-datasets-in-fastai/, 2019.

Radford M Neal. Bayesian learning for neural networks, volume 118. Springer Science & Business Media, 2012.

Andrei Paleyes, Mark Pullin, Maren Mahsereci, Neil Lawrence, and Javier González. Emulation of physical processes with emukit. In Second Workshop on Machine Learning and the Physical Sciences, NeurIPS, 2019.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12:2825–2830, 2011.

Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms. In Advances in Neural Information Processing Systems (NeurIPS), pp. 2960–2968, 2012.

Joaquin Vanschoren. Meta-learning. In Automated Machine Learning - Methods, Systems, Challenges, pp. 35–61. Springer Nature, 2019.

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

Figure 4: PrBO on the 1D Branin function. The leftmost column shows the exponential prior. The other columns show the model and the log pseudo-posterior after 0 (RS only), 10, and 20 BO iterations. PrBO forgets the wrong prior on the local optimum and converges to the global optimum.

A Prior Forgetting

In this section, we show that PrBO can recover from a misleading prior, thanks to our predictive model and the $\alpha = \quad$ parameter in the pseudo-posterior computation Eq. 3. As BO progresses, the predictive model becomes more accurate and receives more weight, guiding optimization away from the wrong prior and towards better values of the objective function. Figure 4 shows PrBO on the 1D Branin function with an exponential prior. Columns (b), (c), and (d) show PrBO after $2 + 1 = 2$ initial samples and 0, 10, 20 BO iterations, respectively. After initialization, as shown in Column (b), the pseudo-posterior is nearly identical to the exponential prior and guides PrBO towards the region of the space on the right, which is towards the local optimum. This happens until the predictive model becomes certain there will be no more improvement from sampling that region (Columns (c) and (d)). After that, the predictive model guides the pseudo-posterior towards exploring regions with high uncertainty. Once the global minimum region is found, the pseudo-posterior starts balancing exploiting the global minimum and exploring regions with high uncertainty, as shown in 4d (bottom). Notably, the pseudo-posterior after > 4 falls to 0 in 4d (top), as the predictive model is certain there will be no improvement from sampling the region of the local optimum.

B EI Derivation

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

$$EI_f(x) := \int_{\text{inf}}^{\infty} \max(f - y, 0)\, p(y|x)\, dy = \int_{\text{inf}}^{f} (f - y)\, \frac{p(x|y)p(y)}{p(x)}\, dy. \tag{6}$$

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

$$\int_{\text{inf}}^{f} (f - y)\, p(x|y)p(y)\, dy = g(x) \int_{\text{inf}}^{f} (f - y)\, p(y)\, dy = f \quad g(x) \quad g(x) \int_{\text{inf}}^{f} y\, p(y)\, dy; \tag{7}$$

so that finally

$$EI_f(x) = \frac{f \quad g(x) \quad g(x) \int_{\text{inf}}^{f} y\, p(y)\, dy}{g(x) + (1 - \quad) b(x)} \Big/ \quad + \frac{b(x)}{g(x)}(1 - \quad)^{1}. \tag{8}$$

C 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 to $= 10$ and the model quantile to $= 0.05$, see Appendices J and I. Before the main BO loop, PrBO is initialized by random sampling $2D + 1$ points from the prior, where D is the number of input dimensions. We use the public implementation of Spearmint[3], which 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 H. 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 10% chance.

We optimize EI 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 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. 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. 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.

XGBoost. 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).

D SVM Regret Comparison

In addition to the experiments in Section 4.2, we show the performance of PrBO on the SVM benchmark. Figure 5 shows a log regret comparison of PrBO, Spearmint, Prior Sampling and 10,000 RS. We note that the results are similar to the other benchmarks in Figure 3. Namely, PrBO

[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	$[e^{-10}; e^{10}]$	X
		$[e^{-10}; e^{10}]$	X
FCNet	learning rate	$[10^{-6}; 10^{-1}]$	X
	batch size	$[8; 128]$	X
	units layer 1	$[16; 512]$	X
	units layer 2	$[16; 512]$	X
	dropout rate l1	$[0:0; 0:99]$	-
	dropout rate l2	$[0:0; 0:99]$	-
XGBoost	learning rate	$[10^{-6}; 10^{-1}]$	X
	gamma	$[0; 2]$	-
	L1 regularization	$[10^{-5}; 10^{3}]$	X
	L2 regularization	$[10^{-5}; 10^{3}]$	X
	number of estimators	$[10; 500]$	-
	subsampling	$[0:1; 1]$	-
	maximum depth	$[1; 15]$	-
	minimum child weight	$[0; 20]$	-

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

with a strong prior outperforms RS and spearmint. PrBO also outperforms Spearmint with a weak prior and even with a uniform prior.

E Spatial Real-world Application

Spatial (Koeplinger et al., 2018) is a programming language and corresponding compiler for the design of application accelerators on recon gurable architectures, e.g. eld-programmable gate arrays (FPGAs). These recon gurable architectures are a type of logic chip that can be recon gured 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 nal 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-speci c basis.

In our experiments, we introduce for the rst 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

Table 2: Search space, priors, and expert con guration 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, 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, 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, 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, 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, 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]

Figure 6: Log regret comparison of RS, HyperMapper, PrBO, and manual optimization on Spatial (mean +/- one std on 5 repetitions). Vertical lines are initialization.

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 nd 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 RF surrogate instead of a GP as explained in Appendix C. These parameters are application speci c and control how much of the FPGAs' resources we want to use to parallelize each step of the application's computation. The goal is to nd which steps are more important to parallelize in order to achieve the fastest 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 con gurations are infeasible. A con guration is considered infeasible if the nal 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 t in the FPGA. To handle these constraints, we use our cBO implementation (Appendix C). Our goal is thus to nd the design with the fastest runtime under the constraint that the design ts the FPGA resource budget.

Table 3: Search space, priors, and expert con guration 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 con guration 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, 0.005, 0.005, 0.1]
x276	Categorical	[false,true]	true	[0.1, 0.9]

Figure 7: Log regret comparison of PrBO with multivariate and univariate KDE priors (mean +/- one std on 5 repetitions). We run the benchmarks for100D iterations, capped at 300.

We compare the performance of PrBO to RS, manual optimization, and HyperMapper (Nardi et al., 2019), the current state-of-the-art BO solution for Spatial. For a fair comparison, both PrBO and HyperMapper use RFs as predictive probabilistic model. The priors for 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 con guration, which ensures all methods start with at least one feasible con guration. The priors, expert con guration, and default con gurations 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 expert con guration for each application are presented in Tables 2, 3, and 4.

Figure 6 shows the log regret on the Spatial benchmarks. PrBO vastly outperforms RS in all benchmarks; notably, RS does not improve over the default con guration in MD Grid. PrBO is also able to leverage the expert's prior and outperform the expert's con guration in all benchmarks (2.6× 1.06×, and10.4× speedup for shallow CNN, deep CNN, and MD Grid, respectively). Compared to HyperMapper, PrBO achieves better performance in the MD Grid benchmark (1.28× speedup). For context, this is a signi cant improvement in the FPGA eld, where a 10% improvement could qualify for acceptance in a top-tier conference. In the CNN benchmarks, PrBO converges to the minima regions faster than HyperMapper (1.5× and1.4× faster for shallow and deep respectively). Thus, PrBO leverages the best of both worlds (the expert's prior knowledge and BO) to provide a new state of the art for Spatial.

F Multivariate Prior Comparison

Figure 7 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 best10D points out of1,000D and10,000,000D 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 de ne these priors without needing additional experiments.

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, 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, around0.2 and0.03 for the weak and strong prior, respectively, whereas the XGBoost benchmark can reach values as high as $f(x) = 600$. Our hypothesis is that this difference comes from the bandwidth estimation ($n^{-\frac{1}{D}}$), which leads to larger bandwidths, consequently, smoother priors, when a multivariate prior is constructed.

Figure 8: Log regret comparison of PrBO and TPE (mean +/- one std on 5 repetitions). We run the benchmarks for $100D$ iterations, capped at 300.

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

Figure 9: Simple regret of sampling from the prior with different priors for our synthetic benchmarks (mean +/- one std on 5 repetitions). A more informative prior gives better results in all benchmarks.

G Comparison to TPE

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

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

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.

We de ne 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 con gurations 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 8 shows a log regret comparison between PrBO and TPE on our four synthetic benchmarks. PrBO 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 con gurations 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-ef cient surrogates, which leads to better performance.

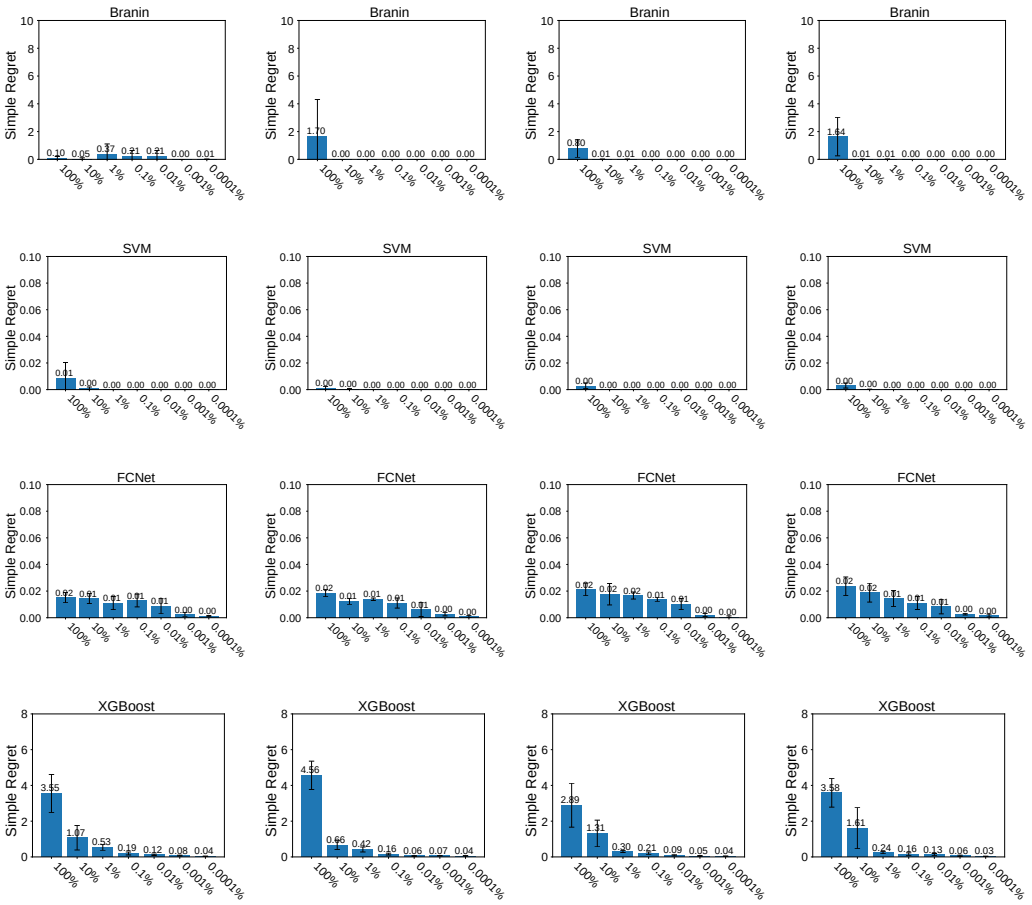
H Prior Bandwidth Selection

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 $n^{\frac{1}{D+b}}$. 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 9 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 nd 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 nd 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 10 shows the performance of PrBO for different priors. The same observations from Figure 9 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 9 and 10 shows that this combination leads to the best results in all benchmarks, as expected.

Figure 11 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



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

Figure 10: Simple regret of PrBO with different priors for our synthetic benchmarks (mean +/- one std on 5 repetitions). A more informative prior gives better results in all benchmarks.

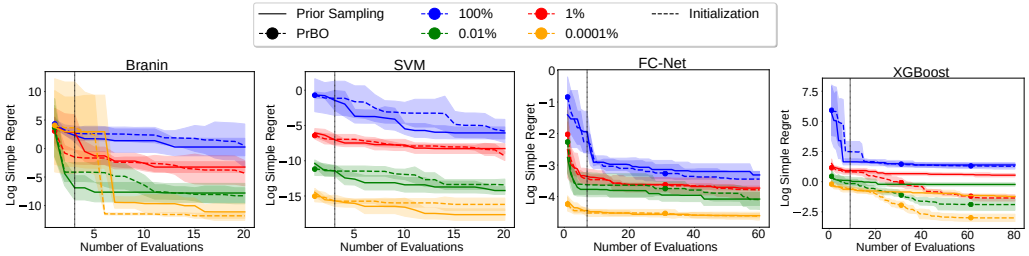


Figure 11: Log simple regret comparison between PrBO and sampling from the prior. The shaded lines are mean +/- one std error.

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