MALIBO: Meta-learning for Likelihood-free Bayesian Optimization

Jiarong Pan [1 2] **Stefan Falkner** [1] **Felix Berkenkamp** [1] **Joaquin Vanschoren** [2]

Abstract

Bayesian optimization (BO) is a popular method to optimize costly black-box functions, and meta-learning has emerged as a way to leverage knowledge from related tasks to optimize new tasks faster. However, existing meta-learning methods for BO rely on surrogate models that are not scalable or are sensitive to varying input scales and noise types across tasks. Moreover, they often overlook the uncertainty associated with task similarity, leading to unreliable task adaptation when a new task differs significantly or has not been sufficiently explored yet. We propose a novel meta-learning BO approach that bypasses the surrogate model and directly learns the utility of queries across tasks. It explicitly models task uncertainty and includes an auxiliary model to enable robust adaptation to new tasks. Extensive experiments show that our method achieves strong performance and outperforms multiple meta-learning BO methods across various benchmarks.

1. Introduction

Bayesian optimization (BO) (Shahriari et al., 2016) is a widely used framework to optimize expensive black-box functions for a wide range of applications, from material design (Frazier & Wang, 2015) to automated machine learning (Hutter et al., 2019). Traditionally, it uses a probabilistic *surrogate model*, often a Gaussian process (GP), to model the black-box function and provide uncertainty estimates that can be used by an *acquisition function* to propose the next query point.

While BO typically focuses on each new target task individually, recent approaches leverage information from previous runs on related tasks through transfer learning (Weiss et al., 2016) and meta-learning (Vanschoren, 2018) to *warm-start* BO. In this context, each *task* denotes the optimization of a specific black-box function and we assume that related tasks share similarities with the target task. For instance, one can warm-start the tuning of a neural network when the same network was previously tuned on related datasets. Previous runs on related tasks can be used to build informed surrogate models (Perrone et al., 2018; Wistuba & Grabocka, 2021; Feurer et al., 2022), restrict the search space (Perrone et al., 2019), or initialize the optimization with configurations that generally score well (Feurer et al., 2014; Volpp et al., 2020).

However, the use of surrogate models also engenders several issues in many of these approaches: (i) GP-based methods scale poorly with the number of observations as well as number of tasks, due to their cubic computational complexity (Rasmussen, 2004). (ii) In practice, observations across tasks can have different scales, e.g., the validation error of an algorithm can be high on one dataset and low on another. Although normalization can be applied to the data from related tasks, normalizing the unseen (target) task data is often challenging, especially when only a few observations are available to estimate its range. Regression-based surrogate models therefore struggle to adequately transfer knowledge from related tasks (Bardenet et al., 2013; Yogatama & Mann, 2014). (iii) While GPs typically assume the observation noise to be Gaussian and homoscedastic, real-world observations often have different noise distributions and can be heteroscedastic. This discrepancy can lead to poor meta-learning and optimization performance (Salinas et al., 2023). Moreover, when adapting to tasks that have limited observations (e.g., early iterations during optimization) or tasks that are significantly different from those seen before, estimating the task similarity becomes challenging due to the scarcity of relevant task information. Hence, it is desirable to explicitly model the uncertainty inherent to such tasks (Finn et al., 2018). Nevertheless, many existing methods warm-start BO by only modeling relations between tasks deterministically (Wistuba et al., 2018; Volpp et al., 2020), making the optimization unreliable.

To tackle these limitations, we propose a novel and scalable meta-learning BO approach[1] that is inspired by the idea

[1]Bosch Center for Artificial Intelligence, Germany [2]Eindhoven University of Technology, Netherlands. Correspondence to: Jiarong Pan <fixed-term.jiarong.pan@de.bosch.com>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

[1]Our code is available in the following repository: https://github.com/boschresearch/meta-learning-likelihood-free-bayesian-optimization

of likelihood-free acquisition function (Tiao et al., 2021; Song et al., 2022). The proposed method overcomes the limitations of surrogate modeling by directly approximating the acquisition function. It makes less stringent assumptions about the observed values, which establishes effective learning across tasks with varying scales and noises. To account for task uncertainty, we introduce a probabilistic meta-learning model to capture the task uncertainty, as well as a novel adaptation procedure based on gradient boosting to robustly adapt to each new task.

This paper makes the following contributions: (i) We propose a scalable and robust meta-learning BO approach that directly models the acquisition function of a given task based on knowledge from related tasks, while being able to cope with heterogeneous observation scales and noise types across tasks. (ii) We use a probabilistic model to meta-learn the task distribution, which enables us to account for the uncertainty inherent in each target task. (iii) We add a novel adaptation procedure to ensure robust adaptation to new tasks that are not well captured by meta-learning.

2. Related Work

Meta-learning Bayesian optimization Various methods have been proposed to improve the data-efficiency of BO through meta-learning and have shown effectiveness in diverse applications (Andrychowicz et al., 2016).

One line of work focuses on the initialization of the optimization (*initial design*) by reducing the search space (Perrone et al., 2019; Li et al., 2022) or reusing promising configurations from similar tasks, where task similarity can be determined using hand-crafted *meta-features* (Feurer et al., 2014) or learned through neural networks (NNs) (Kim et al., 2017). One can also estimate the utility of a configuration using heuristics (Wistuba et al., 2015) or learning-based techniques (Volpp et al., 2020; Hsieh et al., 2021; Maraval et al., 2023). Transfer learning is also employed to modify the surrogate model using multi-task GPs (Swersky et al., 2013; Tighineanu et al., 2022; 2024), additive GP models (Golovin et al., 2017), weighted combinations of independent GPs (Wistuba et al., 2018; Feurer et al., 2022), shared feature representation learned across tasks (Perrone et al., 2018; Wistuba & Grabocka, 2021; Khazi et al., 2023) or pre-training surrogate models on large amount of diverse data (Chen et al., 2022; Müller et al., 2023).

Several methods *simultaneously* learn the initial design and modify the surrogate model. BOHAMIANN (Springenberg et al., 2016) adopts a Bayesian NN as the surrogate model, which is computationally expensive and hard to train. ABLR (Perrone et al., 2018) and BANNER (Berkenkamp et al., 2021) combine a NN to learn a shared feature representation across tasks and task-specific Bayesian linear regression (BLR) layers for scalable adaptation. While ABLR adapts to new tasks by fine-tuning the whole network, BANNER meta-learns a task-independent mean function and only fine-tunes the BLR layer during optimization. However, both methods are sensitive to changes in scale and noise across tasks. To address this, GC3P (Salinas et al., 2020) transforms the observed values via quantile transformation and fits a NN across all related tasks. Although GC3P warm-starts the optimization by using a NN to predict the mean for a GP on the target task, its scalability is limited by its GP surrogate.

Likelihood-free acquisition functions Bayesian optimization does not require an explicit model of the likelihood of the observed values (Garnett, 2022) and can be done by directly approximating the acquisition function. The tree-structured Parzen estimator (TPE) (Bergstra et al., 2011) phrases BO as a density ratio estimation problem (Sugiyama et al., 2012) and uses the density ratio over 'good' and 'bad' configurations as an acquisition function. BORE (Tiao et al., 2021) estimates the density ratio through class probability estimation (Qin, 1998), which is equivalent to modeling the acquisition function with a binary classifier and can be parallellized (Oliveira et al., 2022). By transforming the acquisition function into a variational problem, likelihood-free Bayesian optimization (LFBO) (Song et al., 2022) uses the probabilistic predictions of a classifier to directly approximate the acquisition function. In this paper, we leverage the flexibility of likelihood-free acquisition functions and combine it with a meta-learning model to obtain a sample-efficient, scalable, and robust BO method.

3. Background

Meta-learning Bayesian optimization BO aims to minimize a target black-box function $f : \mathcal{X} \to \mathbb{R}$ over $\mathbf{x} \in \mathcal{X}$. In the case of meta-learning, T related black-box functions $\{f^t(\cdot)\}_{t=1}^T$ are given in advance, each with the same domain $\mathcal{X}$. The optimization is warm-started with previous evaluations on the related functions, $\mathcal{D}^{\text{meta}} = \{\mathcal{D}^t\}_{t=1}^T$ with $\mathcal{D}^t = \{(\mathbf{x}_i^t, y_i^t)\}_{i=1}^{N^t}$, where $y_i^t = f^t(\mathbf{x}_i^t) + \epsilon^t$ are evaluations corrupted by noise ϵ^t and $N^t = |\mathcal{D}^t|$ is the number of observations collected from task f^t. Given a new task at step $N+1$, BO proposes $\mathbf{x}_{N+1}$ and obtains a noisy observation from the target function $y_{N+1} = f(\mathbf{x}_{N+1}) + \epsilon$, with ϵ drawn i.i.d. from some distribution p_ϵ. To obtain the proposal $\mathbf{x}_{N+1}$, a probabilistic surrogate model is first fitted on N previous observations on the target function $\mathcal{D}_N = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$ and the related functions $\mathcal{D}^{\text{meta}}$. For simplicity, we denote $\mathcal{D} := \mathcal{D}_N \cup \mathcal{D}^{\text{meta}}$. The resulting model is used to compute an acquisition function, such as, the expected utility of a given query $\mathbf{x}$,

$$\alpha^{\text{U}}(\mathbf{x}; \mathcal{D}, \tau) = \mathbb{E}_{y \sim p(y|\mathbf{x}, \mathcal{D})}[U(y; \tau)]\,, \tag{1}$$

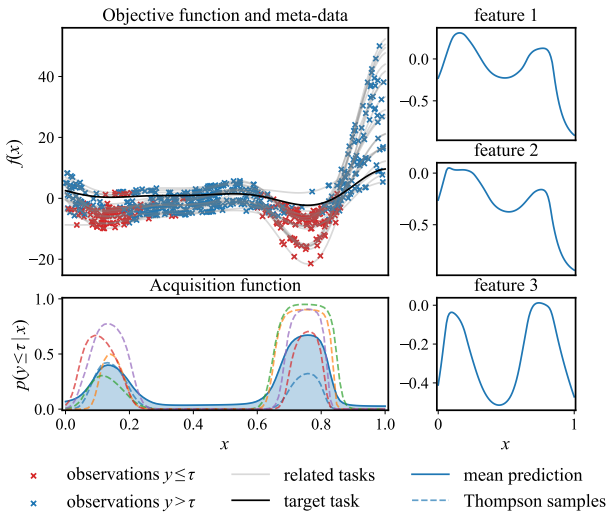


Figure 1. Meta-learning the acquisition function. Left: The top panel shows observations from 10 related tasks and the target task. The top performing observations ($\tau = \Phi^{-1}(\gamma), \gamma = 1/3$) in each task are shown in red, the rest in blue. The bottom panel shows the maximum-a-posteriori estimate of the acquisition function in solid blue while the Thompson samples are shown as dashed curves. Right: Features learned by our model. MALIBO successfully identifies the promising areas in the input space, while the Thompson samples show variability in the meta-learned acquisition function.

where $U(y;\tau)$ is a chosen utility function with a threshold τ that decides the utility of observing y at $\mathbf{x}$ and controls the exploration-exploitation trade-off. The predictive distribution $p(y \mid \mathbf{x}, \mathcal{D})$ is given by the probabilistic surrogate model and the maximizer $\mathbf{x}_{N+1} = \arg\max_{\mathbf{x}\in\mathcal{X}} \alpha(\mathbf{x}; \mathcal{D}, \tau)$ is the proposed candidate. Acquisition functions that take the form of Equation (1) include Expected Improvement (EI) (Močkus, 1975) and Probability of Improvement (PI) (Kushner, 1964). Many others exist, such as UCB (Srinivas et al., 2010), Entropy Search (Hennig & Schuler, 2012; Hernández-Lobato et al., 2014; Wang & Jegelka, 2017) and Knowledge Gradient (Frazier et al., 2009).

Likelihood-free acquisition functions Likelihood-free acquisition functions model the utility of a query without explicitly modeling the predictive distribution. For example, tree-structured Parzen estimators (TPE) (Bergstra et al., 2011) dismiss the surrogate for the outcomes and instead model two densities that split the observations w.r.t. a threshold τ, namely $\ell(\mathbf{x}) = p(\mathbf{x} \mid y \leq \tau, \mathcal{D}_N)$ and $g(\mathbf{x}) = p(\mathbf{x} \mid y > \tau, \mathcal{D}_N)$ for the promising and non-promising data distributions, respectively. The threshold τ relates to the γ-th quantile of the observed outcomes via $\gamma = \Phi(\tau) := p(y \leq \tau)$. In fact, the resulting density ratio (DR) $\alpha^{\text{DR}}(\mathbf{x}; \mathcal{D}_N, \tau) = \ell(\mathbf{x})/g(\mathbf{x})$ is shown to have the same maximum as PI (Song et al., 2022; Garnett, 2022).

BORE (Tiao et al., 2021) improves several aspects of TPE by directly estimating the density ratio instead of solving the more challenging problem of modeling two independent densities as an intermediate step. It rephrases the density ratio estimation as a binary classification problem where all observations within the same class have the same importance. Specifically, they show $\alpha^{\text{DR}}(\mathbf{x}; \mathcal{D}_N, \tau) \propto C_{\boldsymbol{\theta}}(\mathbf{x}) = p(k = 1 \mid \mathbf{x}, D_N, \tau)$, where $k = \mathbb{1}(y \leq \tau)$ represents the binary class labels for classification and the classifier $C_{\boldsymbol{\theta}}$ has learnable parameters $\boldsymbol{\theta}$.

Likelihood-free Bayesian optimization (LFBO) (Song et al., 2022) directly learns an acquisition function in the form of Equation (1) through a classifier. By rephrasing the integral as a variational problem, LFBO involves solving a weighted classification problem with noisy labels for the class $k = 1$, where the weights correspond to utilities. It is shown that the EI acquisition function, where $U(y;\tau) := \max(\tau - y, 0)$, can be estimated by a classifier that optimizes the following objective:

$$\mathcal{L}^{\text{LFBO}}(\boldsymbol{\theta}; \mathcal{D}_N, \tau) = \\ - \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}_N}\left[\max(\tau - y, 0) \ln C_{\boldsymbol{\theta}}(\mathbf{x}) + \ln(1 - C_{\boldsymbol{\theta}}(\mathbf{x}))\right]. \tag{2}$$

The resulting classifier splits promising and non-promising configurations with probabilistic predictions that can be interpreted as the utility of queries, leading to scale-invariant models without noise assumptions and allowing the application of any classification methods (Song et al., 2022). Further details of the algorithms are provided in Appendix A.

4. Methodology

In this section, we introduce our MetA-learning for LIkelihood-free BO (MALIBO) method, which extends LFBO with an effective meta-learning approach. An illustration of our method on a one-dimensional problem is shown in Figure 1. Our approach uses a neural network to meta-learn both a task-agnostic model based on features learned across tasks (right panel in Figure 1), and a task-specific component that provides uncertainty estimation to adapt to new tasks. Additionally, we use Thompson sampling (dashed lines in Figure 1) as an exploratory strategy to account for the task uncertainty. Finally, a residual prediction model (see below) is added to adapt to tasks that are not well captured by the meta-learned model.

Network structure MALIBO uses a structured neural network that combines a meta-learned, task-agnostic model with a task-specific layer. We show an overview in Figure 2 and provide details for the choices below. Following previous works (Perrone et al., 2018; Berkenkamp et al., 2021), our meta-learning model uses a deterministic, task-agnostic model to map the input into features $\boldsymbol{\Phi} = \phi(\mathbf{x})$, where

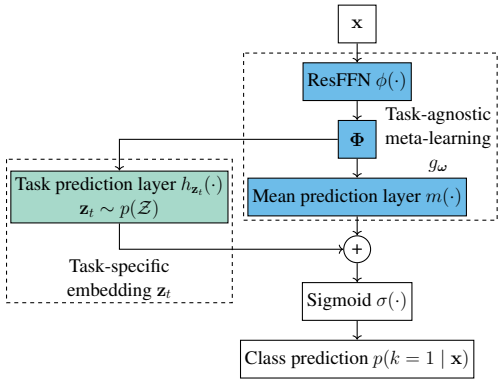


Figure 2. Schematic representation of our meta-learning classifier. A residual feedfoward network (ResFFN) maps the input $\mathbf{x}$ via a shared feature mapping function ϕ. From this, we construct a task-agnostic mean prediction $m(\boldsymbol{\Phi})$ and a task embedding $\mathbf{z}_t$, which is distributed according to a prior distribution $p(\mathcal{Z})$. The feature mapping function ϕ and mean prediction layer m are fixed after meta-training, denoted by the task-agnostic component $g_{\boldsymbol{\omega}}$. Finally, we add and convert them to a class prediction via the sigmoid function.

$\phi : \mathcal{X} \to \mathbb{R}^d$ is a learnable feature mapping shared across all tasks and d is the predefined dimensionality of the feature space. We use a residual feedforward network (ResFFN) for learning ϕ, which has been shown to be robust to network hyperparameters and generalizes well to different problems (Huang et al., 2020). To enable our model to provide good initial proposals, we introduce a task-agnostic mean prediction layer $m : \mathbb{R}^d \to \mathbb{R}$ that learns the promising areas from the related tasks. We refer to the combined task-agnostic components m and ϕ as $g_{\boldsymbol{\omega}}$ (shown in blue), which is parameterized by $\boldsymbol{\omega}$. To allow adaptation on each task t, we use a task prediction layer $h_{\mathbf{z}_t} : \mathbb{R}^d \to \mathbb{R}$, which is parameterized by layer weights $\mathbf{z}_t \in \mathcal{Z} \subseteq \mathbb{R}^d$. Since each $\mathbf{z}_t$ embeds in a low dimensional latent space $\mathcal{Z}$ and is a unique vector for each task, we refer to $\mathbf{z}_t$ as the task embedding. We will train our model such that the $\{\mathbf{z}_t\}_{t=1}^T$ follow a known distribution $p(\mathcal{Z})$ and discuss below how to use this as a prior for target task adaptation. Lastly, in order to obtain classification outputs as in LFBO, we apply the sigmoid function to produce probabilistic class predictions $p(k = 1 \mid \mathbf{x})$. The prediction for an observation in task t is then given by $C(\mathbf{x}_t) = \sigma(m(\phi(\mathbf{x})) + h_{\mathbf{z}_t}(\phi(\mathbf{x}))) = \sigma(m(\boldsymbol{\Phi}) + \mathbf{z}_t^\mathsf{T}\boldsymbol{\Phi})$.

Meta-learning Directly optimizing $\mathcal{L}^{\text{LFBO}}$ to meta-learn our model would lead to task embeddings that do not conform to any particular prior task distribution $p(\mathcal{Z})$, and thus render task adaptation difficult and unreliable (Finn et al., 2018). Therefore, we regularize the task embeddings $\{\mathbf{z}_t\}_{t=1}^T$ during training to enable Bayesian inference. In addition, such regularization can also avoid overfitting in the task space $\mathcal{Z}$ and improves the generalization performance of our model. Specifically, we assume the prior of the task embeddings to be a multivariate normal (MVN), $p(\mathcal{Z}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$ and apply a regularization term to bring the empirical distribution of the $\{\mathbf{z}_t\}_{t=1}^T$ close to the prior distribution. The loss used for training on the meta-data reads:

$$\mathcal{L}^{\text{meta}}(\boldsymbol{\omega}, \{\mathbf{z}_t\}_{t=1}^T) = \frac{1}{T}\sum_{t=1}^{T} \mathcal{L}^{\text{LFBO}}(\boldsymbol{\omega}, \mathbf{z}_t; \mathcal{D}^t, \tau) + \lambda \mathcal{R}(\{\mathbf{z}_t\}_{t=1}^T; p(\mathcal{Z}))\,, \quad (3)$$

where the first term is the loss function from LFBO as in Equation (2), weighting the observations in the meta-data with improvements and the second term $\mathcal{R}$ is the regularization term weighted by λ. We regularize the empirical distribution of $\{\mathbf{z}_t\}_{t=1}^T$ to match the Gaussian prior in a tractable way (Saseendran et al., 2021):

$$\mathcal{R}(\{\mathbf{z}_t\}_{t=1}^T; p(\mathcal{Z})) = \lambda_{\text{KS}} \sum_{j=1}^{d} (F([\mathbf{z}_t]_j) - \Phi([\mathbf{z}_t]_j))^2 + \lambda_{\text{Cov}} \|\mathbf{I} - \text{Cov}(\{\mathbf{z}_t\}_{t=1}^T)\|_{\text{F}}^2\,, \quad (4)$$

where the first term matches the marginal cumulative distribution functions (CDFs) similar to a Kolmogrov-Smirnov (KS) test, and the second term matches the empirical covariance of the task embeddings to the covariance of the prior. The hyperparameters λ_{KS} and λ_{Cov} encode the trade-off between these two terms. We denote F as the empirical CDF and Cov as the empirical covariance matrix. For more details we refer to Appendix C.

We only consider a uni-modal Gaussian prior in this work, as we will show it already demonstrates strong performance against other baselines. For more complex task distributions, one could extend it with multi-modal Gaussian prior (Saseendran et al., 2021).

Task adaptation After meta-training, the model can adapt to new tasks by estimating an embedding $\mathbf{z}$ based on the learned feature mapping function ϕ. In principle, one could use a maximum likelihood classifier obtained by directly optimizing Equation (2) w.r.t. $\mathbf{z}$. However, such a classifier does not consider the task uncertainty and would suffer from unreliable adaptation (Finn et al., 2018) and over-exploitation (Oliveira et al., 2022). Furthermore, when a potential disparity between the distribution of the meta-data and the non-i.i.d. data collected during optimization arises, a probabilistic model would be informed via uncertainty estimation and thereby can exploit the meta-learned knowledge less. Therefore, we propose to use a Bayesian approach for task adaptation, which makes our classifier uncertainty-aware and more exploratory.

Consider the task embedding $\mathbf{z}$ for the target task follows a distribution $p(\mathbf{z} \mid \mathcal{D}_N)$ after N observations, then the

predictive distribution of our model can be written as

$$C(\mathbf{x}; \boldsymbol{\omega}, \mathcal{D}_N) = \int p(k = 1 \mid \boldsymbol{\omega}, \mathbf{z}) p(\mathbf{z} \mid \mathcal{D}_N) \, \mathrm{d}\mathbf{z} \,, \quad (5)$$

which accounts for the epistemic uncertainty in the task embedding. Since the parameters $\boldsymbol{\omega}$ of task-agnostic model $g_{\boldsymbol{\omega}}$ are fixed after meta-training, we denote our classifier as $C(\mathbf{x})$ for simplicity.

As there is no analytical way to evaluate the integration in Equation (5), we have to resort to approximation methods, such as Laplace approximation (Bishop & Nasrabadi, 2006), variational inference (Graves, 2011), and Markov chain Monte Carlo (Homan & Gelman, 2014). We consider Laplace approximation for $p(\mathbf{z} \mid \mathcal{D}_N)$ as a fast and scalable method, and show its competitive performance against other, more expensive alternatives in Appendix D.4.

Laplace's method fits a Gaussian distribution around the maximum-a-posteriori (MAP) estimate of the distribution and matches the second order derivative at the optimum. In the first step, we obtain the MAP estimate by maximizing the posterior of our classifier C parameterized by $\mathbf{z}$. To be consistent with the regularization used during meta-training, we use a standard, isotropic Gaussian prior for the weights: $p(\mathbf{z}) = \mathcal{N}(\mathbf{z} \mid \mathbf{0}, \mathbf{I})$. Given observations $\mathcal{D}_N$, the negative log posterior $p(\mathbf{z} \mid \mathcal{D}_N)$ is proportional to

$$\mathcal{L}^{\text{MALIBO}}(\mathbf{z}) = \frac{1}{2}\mathbf{z}^\mathsf{T}\mathbf{z} - \sum_{n=1}^{N} \left(k_n(\tau - y) \ln \hat{k}_n + \ln(1 - \hat{k}_n) \right) , \quad (6)$$

where $\hat{k} = \sigma(m(\boldsymbol{\Phi}) + \mathbf{z}^\mathsf{T}\boldsymbol{\Phi})$ is the class probability prediction and the MAP estimate of the weights given by $\mathbf{z}_{\text{MAP}} = \arg\min_{\mathbf{z} \in \mathcal{Z}} \mathcal{L}^{\text{MALIBO}}$. As a second step, we compute the negative Hessian of the log posterior

$$\boldsymbol{\Sigma}_N^{-1} = \boldsymbol{\Sigma}_0^{-1} + \sum_{n=1}^{N} (k_n(\tau - y) + 1) \hat{k}_n (1 - \hat{k}_n) \boldsymbol{\Phi}_n \boldsymbol{\Phi}_n^\mathsf{T} , \quad (7)$$

which serves as the precision matrix for the approximated posterior $q(\mathbf{z}) = \mathcal{N}(\mathbf{z} \mid \mathbf{z}_{\text{MAP}}, \boldsymbol{\Sigma}_N)$. Therefore Equation (5) can be approximated as

$$C(\mathbf{x}) \simeq \int p(k = 1 \mid \boldsymbol{\omega}, \mathbf{z}) q(\mathbf{z}) \, \mathrm{d}\mathbf{z} \,. \quad (8)$$

Having developed a meta-learning model, we now focus on how to utilize this model to encourage exploration and ensure reliable task adaptation.

Uncertainty-based exploration In the early phase of optimization, every meta-learning model has to reason about the target task properties based only on the limited data available, which can lead to highly biased results and over-exploitation (Finn et al., 2018). Moreover, LFBO also suffers from similar issue even without meta-learning (Song et al., 2022). Therefore, we propose to use Thompson sampling based on task uncertainty for constructing a more exploratory acquisition function, and the resulting sampled predictions is generated by

$$\hat{C}(\mathbf{x}) = \sigma \left(m(\phi(\mathbf{x})) + h_{\hat{\mathbf{z}}}(\phi(\mathbf{x})) \right) , \quad \hat{\mathbf{z}} \sim q(\mathbf{z}) \,. \quad (9)$$

Besides stronger exploration in the early phases of optimization, Thompson sampling also enables us to extend MALIBO to parallel BO by using multiple Thompson samples of the acquisition function in parallel. It is shown that this bypasses the sequential scheme of traditional BO, without introducing the common computational burden of more sophisticated methods (Kandasamy et al., 2018). We briefly explore this strategy in Appendix F.

Gradient boosting as a residual prediction model Operating in a meta-learned feature space enables fast task adaptation for our Bayesian classifier. However, it relies on the assumption that the meta-data is sufficient and representative for the task distribution, which does not always hold in practice. Moreover, a distribution mismatch between observations $\mathcal{D}_N$ and meta-data $\mathcal{D}^{\text{meta}}$ can arise when $\mathcal{D}_N$ is generated by an optimization process while $\mathcal{D}^{\text{meta}}$ consists of, e.g., i.i.d. samples.

We employ a residual model independent of the meta-learning model, such that, even given non-informative features, our classifier is able to regress to an optimizer that operates in the input space $\mathcal{X}$. We propose to use gradient boosting (GB) (Friedman, 2001) as a residual prediction model for classification, which consists of an ensemble of weak learners that are sequentially trained to correct the errors from the previous ones. Specifically, we replace the first weak learner by a strong learner, i.e., our meta-learned classifier. With Thompson sampling, our classifier can be written as

$$C_{\text{GB}}(\mathbf{x}) = \sigma \left(m(\phi(\mathbf{x})) + h_{\hat{\mathbf{z}}}(\phi(\mathbf{x})) + \sum_{i=1}^{M} r_i(\mathbf{x}) \right) , \quad (10)$$

where each r_i represents the i-th trained base-learner for the error correction from gradient boosting. In addition to robust task adaptation, this approach offers two advantages: First, gradient boosting does not require an additional weighting scheme for combining different classifiers and automatically determines the weight of the meta-learned model; Second, gradient boosting demonstrates strong performance for LFBO on various benchmarks (Song et al., 2022), which makes our classifier achieve competitive performance even when meta-learning fails, as shown in Appendix D.3. The resulting residual model is trained solely

Algorithm 1 MALIBO: Meta-learning for likelihood-free Bayesian optimization

Meta-learning:
Input: $\mathcal{D}^{\text{meta}} = \{\mathcal{D}^t\}_{t=1}^T$, proportion $\gamma \in (0, 1)$
$k = \mathbb{1}(y \leq \tau)$, where $\tau = \Phi^{-1}(\gamma)$
/* generate binary labels */
$g_{\boldsymbol{\omega}} \leftarrow \arg\min_{\boldsymbol{\omega}} \mathcal{L}^{\text{meta}}$; // Equation (3)
Bayesian optimization with meta-learning:
Input: Fixed $g_{\boldsymbol{\omega}}$ after meta-leaning
$\mathbf{x}_0 \leftarrow \arg\max_{\mathbf{x}} g_{\boldsymbol{\omega}}(\mathbf{x})$
$\mathcal{D} \leftarrow \{(\mathbf{x}_0, f(\boldsymbol{x}_0) + \epsilon)\}$
while *has budget* **do**
  $\mathbf{z}_{\text{MAP}} \leftarrow \arg\min_{\mathbf{z}} \mathcal{L}^{\text{MALIBO}}$; // Equation (6)
  Update precision matrix $\boldsymbol{\Sigma}_N^{-1}$; // Equation (7)
  $\hat{\mathbf{z}} \sim \text{MVN}(\mathbf{z}_{\text{MAP}}, \boldsymbol{\Sigma}_N)$
  $\mathbf{x}_* \leftarrow \arg\max_{\mathbf{x}} C_{\text{GB}}(\mathbf{x}; \hat{\mathbf{z}})$; // Equation (10)
  $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{x}_*, f(\mathbf{x}_*) + \epsilon)\}$
end while

on the target task data and thus might overfit in the early iterations with limited data. To avoid this, we apply gradient boosting only after a few iterations of Thompson sampling exploration and train it with early stopping. Note that this does not diminish the usefulness of the residual model, because our goal is to encourage exploration in early iterations as outlined in Section 4, and gradually rely more on the knowledge from the target task. We refer to Appendix G for more implementation details.

5. Experiments

In this section, we first show the effects of using Thompson sampling and gradient boosting through a preliminary ablation study. Subsequently, we describe the experiments conducted to empirically evaluate our method. For the choice of problems, we focus on automated machine learning (AutoML), i.e., hyperparameter optimization (HPO) and neural architecture search (NAS). To highlight the time efficiency of our proposed method, we include a runtime analysis. Additionally, a quantitative ablation study is presented to assess the impact of various components within our framework. Lastly, we evaluate our method on synthetic functions with multiplicative noise to study robustness towards data with heterogeneous scale and noise,

Baselines We compare our method against multiple baselines across all problems. As methods without meta-learning, we pick random search (Bergstra & Bengio, 2012), LFBO (Song et al., 2022) and Gaussian process (GP) (Snoek et al., 2012)) for our experiments. For meta-learning BO methods, we choose ABLR (Perrone et al., 2018), BaNNER (Berkenkamp et al., 2021), RGPE (Feurer et al., 2022),

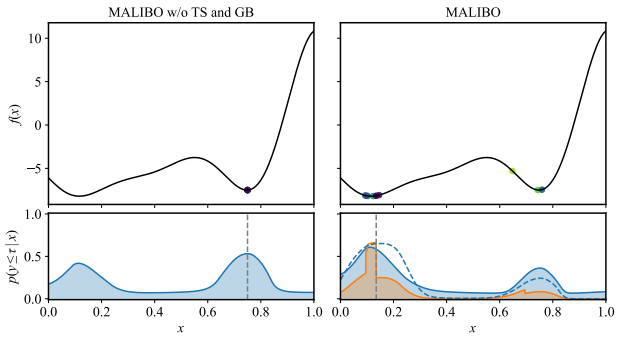


Figure 3. Effects of exploration and residual predictions. Color circles denote the optimization queries (from bright to dark), the dashed curve denotes a Thompson sample (TS) of the acquisition function and the orange curve shows the sample combined with gradient boosting (GB).

GC3P (Salinas et al., 2020), FSBO (Wistuba & Grabocka, 2021), MetaBO (Volpp et al., 2020), PFN (Müller et al., 2023) and DRE (Khazi et al., 2023) as representative algorithms. Additionally, we consider a simple baseline for extending LFBO with meta-learning, called LFBO+BB, which combines LFBO with bounding-box search space pruning (Perrone et al., 2019) as a meta-learning approach. For all LFBO-based methods, including MALIBO, we set the required threshold $\gamma = 1/3$ following Song et al. (2022).

Evaluation metrics In order to aggregate performances across tasks, we use *normalized regret* as the quantitative performance measure for AutoML problems (Wistuba et al., 2018). This is defined as $\min_{\mathbf{x} \in \mathcal{X}_N}(f^t(\mathbf{x}) - f^t_{\text{min}})/(f^t_{\text{max}} - f^t_{\text{min}})$, where $\mathcal{X}_N$ denotes the set of inputs that have been selected by an optimizer up to iteration N, f^t_{min} and f^t_{max} respectively represent the minimum and the maximum objective computed across all offline evaluations available for task t. We report the mean normalized regret across all tasks within a benchmark as the aggregated result. For all benchmarks, we report the results by mean and standard error across *100 random runs*.

Effects of exploration and residual prediction We first investigate the effect of Thompson sampling and the residual prediction model when optimizing a Forrester function (Sobester et al., 2008) as a toy example. By using the meta-learned model as shown in Figure 1, MALIBO performs task adaptation on a new Forrester function for 10 iterations. We compare the results of MALIBO against a variant without the proposed Thompson sampling and gradient boosting, which only uses the approximated posterior predictive distribution in Equation (8) by probit approximation (Bishop & Nasrabadi, 2006) for the acquisition function. As shown in Figure 3, MALIBO without Thompson sampling fails to adapt the new task with little exploration and optimizes

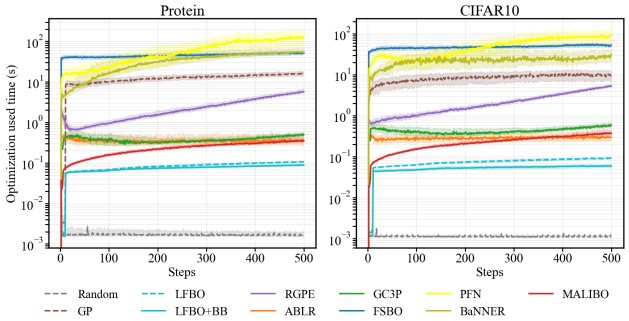


Figure 4. Runtime of different BO algorithms over optimization steps. We show the typical results for two benchmarks and plot the medial inter-quantiles to remove outliers.

greedily around the local optimum. This greedy optimization occurs due to the strong dependence of LFBO on a good initialization to not over-exploit. In contrast, the proposed MALIBO allows the queries to cover both possible optima by encouraging explorations. In addition, gradient boosting performs the refinement beyond the smooth meta-learned acquisition function, which can be seen in the discontinuity in the predictions. By suppressing the predicted utility in the less promising areas, gradient boosting refines the acquisition function to focus on the lower value region. We provide an extensive ablation study on the effects of different components in MALIBO and refer to Appendix D for more details.

Runtime analysis To confirm the scalability of MALIBO, we compare the runtime between methods, specifically the time required for the algorithm to propose a new candidate. As shown in Figure 4, the introduction of latent features and the Laplace approximation only adds negligible overhead compared to LFBO, while MALIBO's runtime increases slowly with the number of observations. In contrast, all other meta-learning methods, except for LFBO+BB, are considerably slower than MALIBO. We include more detailed experimental results in Appendix E.2.

Real-world benchmarks We empirically evaluate our method on various real-world optimization tasks, focusing on AutoML problems, including neural architecture search (NASBench201) (Dong & Yang, 2020), hyperparameter optimization for neural networks (HPOBench) (Klein & Hutter, 2019) and machine learning algorithms (HPO-B) (Pineda-Arango et al., 2021). In NASBench201, we consider designing a neural cell with 6 discrete parameters, totaling $15,625$ unique architectures, evaluated on CIFAR-10, CIFAR-100 (Krizhevsky et al., 2009) and ImageNet-16 (Chrabaszcz et al., 2017). The goal is to find the optimal architecture for a neural network that yields the highest validation accuracy. For HPOBench, the aim is to find the optimal hyperparameters for a two-layer feed-forward regression network on four popular UCI datasets (Dua & Graff, 2017). The search space is 9-dimensional and the optimization objective is the validation mean squared error after training with the corresponding network configuration. In HPO-B, the focus is on optimizing the hyperparameters for different machine learning models to maximize accuracy across various tasks. This benchmark comprises about 6 million evaluations of hyperparameters, across 16 search spaces that correspond to different machine learning models. Each search space varies in dimensionality ranging from 2 to 18 and includes several tasks, which are divided into training, validation and test tasks. Compared to the extensive evaluations in HPO-B, both HPOBench and NASBench201 have significantly fewer related tasks and serve as examples of performance with limited meta-data. We provide details for benchmarks in Appendix H.

To train and evaluate the meta-learning BO methods in HPOBench and NASBench201, we conduct our experiments in a leave-one-task-out way: all meta-learning methods use one task as the target task and all others as related tasks. In this way, every task in a benchmark is picked as the target task once. To construct the meta-datasets, we randomly select 512 configuration-objective pairs from the related tasks, considering the limitations of RGPE in handling large meta-datasets. All meta-learning methods, except MetaBO, are trained from scratch for each independent run, to account for variations due to the randomly sampled meta-data. Because of its long training time, MetaBO is trained once for each target problem on more meta-data than other methods to avoid limiting its performance with a bad subsample and we show its results only for HPOBench and NASBench201. As for HPO-B, we utilize the provided meta-train and meta-validation dataset to train the meta-learning methods and evaluate all methods on the meta-test data. While all methods optimize the target tasks from scratch in the other two benchmarks, the first five initial observations in HPO-B is fixed as random seed and therefore we only show the performances starting after the initialization. We refer to Appendix G for more experimental details.

The aggregated results for all three benchmarks are summarized in Figure 5. It is evident that MALIBO consistently achieves strong anytime performance, surpassing other methods that either exhibit poor warm-starting or experience early saturation of performance. Notably, MALIBO outperforms other methods by a large margin in HPOBench, possibly because we focus on minimizing the validation error of a regression model in this benchmark. This task poses a significant challenge for GP-based regression models, as the observation values undergo abrupt changes and have varying scales across tasks, thereby violating the smoothness and noise assumptions inherent in these models. In most

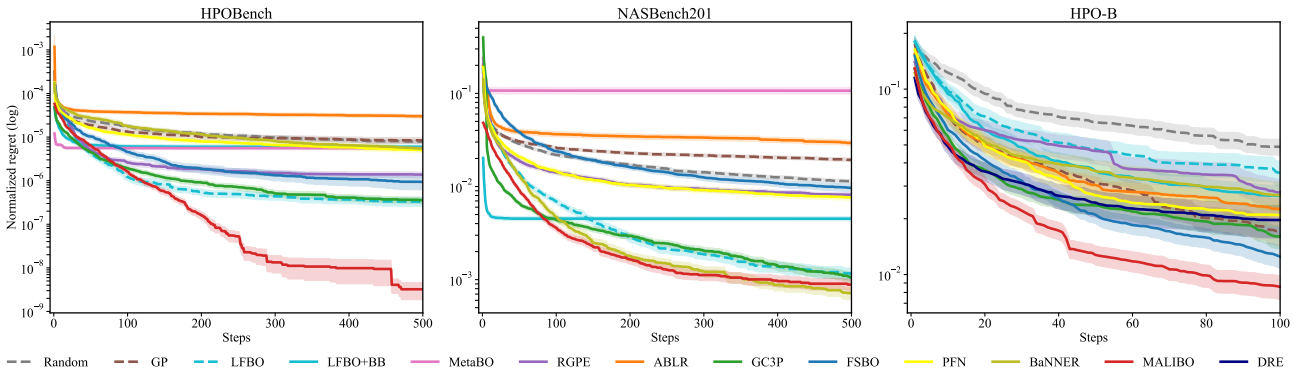


Figure 5. Aggregated normalized regrets for BO algorithms on real-world AutoML problems.

benchmarks, PFN and BaNNER exhibit comparable performance to GP, except in NASBench201. GC3P performs competitively only after the Copula process is fitted and LFBO matches its final performance. LFBO+BB exhibits similar performance as MALIBO in warm-starting and converges quickly, but the search space pruning technique forbids the method to explore regions beyond the promising areas in the meta-data, making its final performance even worse than its non-meta-learning counterpart. ABLR, RGPE and FSBO perform poorly on most of the benchmarks, except for HPO-B, because their meta-learning techniques require more meta-data for effective warm-starting, making them less data-efficient than MALIBO. MetaBO shows strong warm-starting performance in HPOBench while it fails in NASBench201. This is possibly due to the higher diversity in NASBench201 compared to HPOBench, and MetaBO fails to transfer knowledge from tasks that are significantly different from the target task. The poor task adaptation ability of MetaBO is also found by other studies (Wistuba & Grabocka, 2021; Wang et al., 2022). For more experimental results, we refer to Appendix E.3.

Ablation study To understand the impact of each component within MALIBO, we conduct a quantitative ablation study and introduce the following variants:

- MALIBO (Probit): Employs a probit approximation (detailed in Appendix B) for the marginalized form of the acquisition function. This variant does not use gradient boosting.
- MALIBO (TS): Utilizes only Thompson sampling, omitting the gradient boosting component.
- MALIBO (RES): Excludes the mean prediction layer $m(\cdot)$.
- MALIBO (MEAN): Removes the task prediction layer $h_{\mathbf{z}_t}(\cdot)$ and utilizes only the task-agnostic component $g_{\boldsymbol{\omega}}$.
- MALIBO (RF): Substitutes gradient boosting with a random forest (RF) classifier.
- MALIBO (MLP): Replaces gradient boosting with a multi-layer perceptron (MLP) classifier.

The results illustrated in Figure 6 reveal several key insights. Due to the lack of a mean prediction layer, MALIBO (RES) exhibits the poorest warm-starting performance across all benchmarks, potentially leading to worse final performance. Although the mean prediction layer improves initial performance, relying solely on it may result in over-fitting to the meta-data due to insufficient exploration capabilities. As demonstrated by the performance of MALIBO (MEAN), while it achieves the best results among all variants in NASBench201, its performance on the other two benchmarks is inferior to the proposed method. In contrast, variants that include an uncertainty-aware task prediction layer, such as MALIBO (RES) and MALIBO, perform task adaptation more reliably. Although MALIBO (TS) also encourages exploration, the absence of a residual prediction model results in a significant performance decrease when the amount of meta-data is limited, as observed in HPOBench and NASBench201. The comparison of MALIBO (MLP) and MALIBO (RF) highlights the superiority of gradient boosting over other classifiers such as random forest and MLP for the residual prediction model. For more detailed experimental results, refer to Appendix D.6.

Robustness against heterogeneous noise We use synthetic function ensembles (Berkenkamp et al., 2021) to test the robustness against heterogeneous noise in the data. We focus on the Hartmann3D function ensemble (Dixon, 1978), which is a three-dimensional problem with four local minima. Their locations and the global minimum vary across different functions. See Appendix H for more details.

To avoid biasing this experiment towards a single method, we use a heteroscedastic noise incompatible with any assumptions about the noise of any method. In particular,

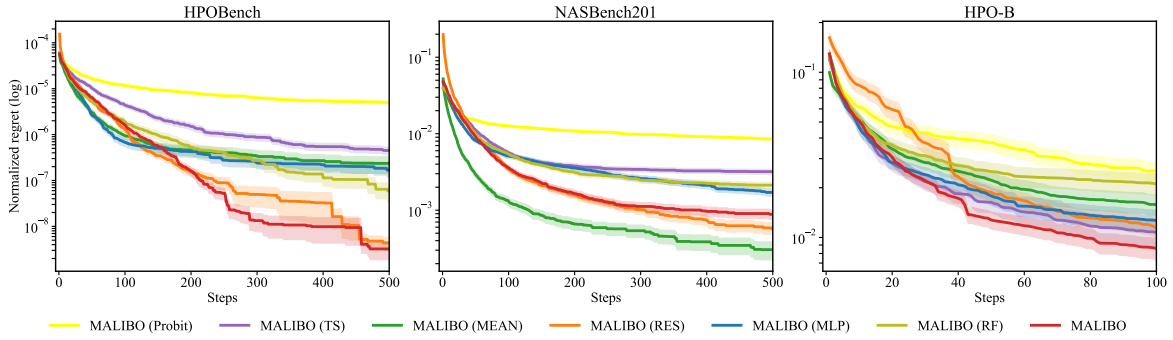


Figure 6. Aggregated normalized regrets of MALIBO variants on real-world AutoML problems.

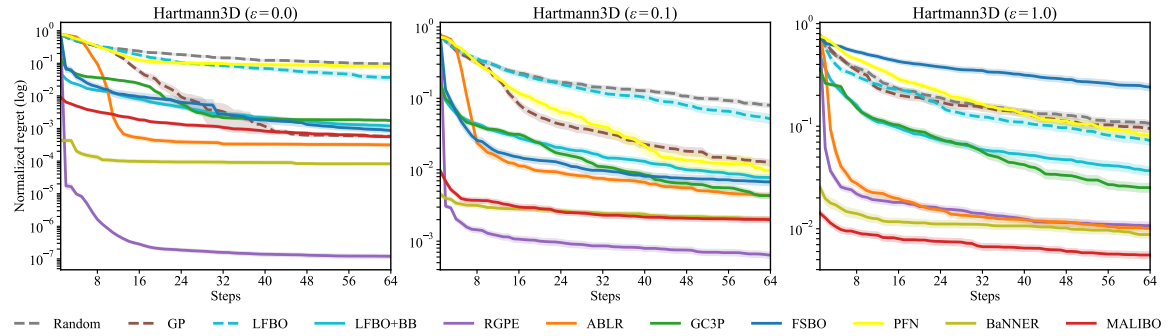


Figure 7. Normalized regret for BO algorithms on Hartmann3D with various levels of multiplicative noise.

this violates the GP methods' and ABLR's assumption of homoscedastic, Gaussian noise. GC3P makes a similar assumption after the nonlinear transformation of the observation values, which does not translate to any well-known noise model. LFBO, LFBO+BB and MALIBO make no explicit noise assumptions, but optimize for the best mean. We choose a multiplicative noise, i.e., $y = f(\mathbf{x}) \cdot (1 + \epsilon \cdot n)$, where $n \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$. The noise corrupts observations with larger values more, while having a smaller effect on those with lower values. To see the robustness with different noise levels, we evaluate $\epsilon \in \{0, 0.1, 1.0\}$. For meta-training, we randomly sample 512 noisy observations from 256 functions in the ensemble. We show our results in Figure 7, where we can see across all noise levels, our method learns a meaningful prior for the optimization. The GP-based methods, despite their strong performance in the noise-free case, especially RGPE, degrade significantly with increasing noise levels.

6. Conclusion

We introduced MetA-learning for LIkelihood-free BO (MALIBO), a method that directly models the acquisition function from observations coupled with meta-learning. This method is computationally efficient and robust to heterogeneous scale and noise across tasks, which poses challenges for other methods. Furthermore, MALIBO enhances data efficiency and incorporates a Bayesian classifier with Thompson sampling to account for task uncertainty, ensuring reliable task adaptation. For robust adaptation to tasks that are not captured by meta-learning, we integrate gradient boosting as a residual prediction model into our framework. Empirical results demonstrate the superior performance of the proposed method across various benchmarks.

Despite promising experimental results, some limitations of the method should be noted. (i) The exploitation and exploration parameter τ in likelihood-free BO algorithms could be treated more carefully, e.g., via a probabilistic treatment (Tiao et al., 2021). (ii) The regularization hyperparameter λ, while robust across our experiments, may lead to suboptimal outcomes in other scenarios. (iii) Using a uni-modal prior could be restrictive for more complex task distributions. Although a generalization to a Gaussian mixture model exists (Saseendran et al., 2021), its efficacy within MALIBO remains unverified.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Andrychowicz, M., Denil, M., Colmenarejo, S. G., Hoffman, M. W., Pfau, D., Schaul, T., Shillingford, B., and de Freitas, N. Learning to learn by gradient descent by gradient descent. In *Neural Information Processing Systems*, pp. 3988–3996, 2016.

Bardenet, R., Brendel, M., Kégl, B., and Sebag, M. Collaborative hyperparameter tuning. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, pp. II–199–II–207, 2013.

Bergstra, J. and Bengio, Y. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13: 281–305, 2012.

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

Berkenkamp, F., Eivazi, A., Grossberger, L., Skubch, K., Spitz, J., Daniel, C., and Falkner, S. Probabilistic meta-learning for bayesian optimization, 2021. URL https://openreview.net/forum?id=fdZvTFn8Yq.

Bishop, C. M. and Nasrabadi, N. M. *Pattern recognition and machine learning*. Springer, 2006.

Byrd, R. H., Lu, P., Nocedal, J., and Zhu, C. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208, 1995.

Chen, Y., Song, X., Lee, C., Wang, Z., Zhang, R., Dohan, D., Kawakami, K., Kochanski, G., Doucet, A., Ranzato, M., et al. Towards learning universal hyperparameter optimizers with transformers. *Advances in Neural Information Processing Systems*, 35:32053–32068, 2022.

Chrabaszcz, P., Loshchilov, I., and Hutter, F. A downsampled variant of imagenet as an alternative to the cifar datasets. *arXiv preprint arXiv:1707.08819*, 2017.

Clevert, D., Unterthiner, T., and Hochreiter, S. Fast and accurate deep network learning by exponential linear units (elus). In *International Conference on Learning Representations*, 2016.

Demšar, J. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7 (1):1–30, 2006.

Dixon, L. C. W. The global optimization problem. an introduction. *Toward global optimization*, 2:1–15, 1978.

Dong, X. and Yang, Y. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations*, 2020.

Dua, D. and Graff, C. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

Feurer, M., Springenberg, J. T., and Hutter, F. Using meta-learning to initialize bayesian optimization of hyperparameters. In *Proceedings of the 2014 International Conference on Meta-Learning and Algorithm Selection - Volume 1201*, pp. 3–10, 2014.

Feurer, M., Letham, B., Hutter, F., and Bakshy, E. Practical transfer learning for bayesian optimization. *arXiv preprint arXiv:1802.02219*, 2022.

Finn, C., Xu, K., and Levine, S. Probabilistic model-agnostic meta-learning. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

Frazier, P., Powell, W., and Dayanik, S. The knowledge-gradient policy for correlated normal beliefs. *INFORMS journal on Computing*, 21(4):599–613, 2009.

Frazier, P. I. and Wang, J. Bayesian optimization for materials design. In *Information science for materials discovery and design*, pp. 45–75. Springer, 2015.

Friedman, J. H. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5):1189 – 1232, 2001.

Garnett, R. *Bayesian Optimization*. Cambridge University Press, 2022.

Golovin, D., Solnik, B., Moitra, S., Kochanski, G., Karro, J., and Sculley, D. Google vizier: A service for black-box optimization. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1487–1495. Association for Computing Machinery, 2017.

Graves, A. Practical variational inference for neural networks. In *Advances in Neural Information Processing Systems*, volume 24, 2011.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

Hennig, P. and Schuler, C. J. Entropy search for information-efficient global optimization. *Journal of Machine Learning Research*, 13(6), 2012.

Hernández-Lobato, J. M., Hoffman, M. W., and Ghahramani, Z. Predictive entropy search for efficient global optimization of black-box functions. *Advances in neural information processing systems*, 27, 2014.

Homan, M. D. and Gelman, A. The no-u-turn sampler: Adaptively setting path lengths in hamiltonian monte carlo. *J. Mach. Learn. Res.*, 15(1):1593–1623, 2014.

Hsieh, B.-J., Hsieh, P.-C., and Liu, X. Reinforced few-shot acquisition function learning for bayesian optimization. In *Advances in Neural Information Processing Systems*, 2021.

Huang, K., Wang, Y., Tao, M., and Zhao, T. Why do deep residual networks generalize better than deep feedforward networks? — a neural tangent kernel perspective. In *Advances in Neural Information Processing Systems*, volume 33, pp. 2698–2709, 2020.

Hutter, F., Kotthoff, L., and Vanschoren, J. (eds.). *Automated Machine Learning - Methods, Systems, Challenges*. Springer, 2019.

Kandasamy, K., Krishnamurthy, A., Schneider, J., and Poczos, B. Parallelised bayesian optimisation via thompson sampling. In *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*, volume 84, pp. 133–142, 2018.

Khazi, A. S., Arango, S. P., and Grabocka, J. Deep ranking ensembles for hyperparameter optimization. In *The Eleventh International Conference on Learning Representations*, 2023.

Kim, J., Kim, S., and Choi, S. Learning to warm-start bayesian hyperparameter optimization. *arXiv preprint arXiv:1710.06219*, 2017.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

Klein, A. and Hutter, F. Tabular benchmarks for joint architecture and hyperparameter optimization. *arXiv preprint arXiv:1905.04970*, 2019.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. *online*, 2009.

Kushner, H. J. 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, 03 1964.

Li, Y., Shen, Y., Jiang, H., Bai, T., Zhang, W., Zhang, C., and Cui, B. Transfer learning based search space design for hyperparameter tuning. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 967–977, 2022.

Maraval, A., Zimmer, M., Grosnit, A., and Bou Ammar, H. End-to-end meta-bayesian optimisation with transformer neural processes. *Advances in Neural Information Processing Systems*, 36, 2023.

Močkus, J. On bayesian methods for seeking the extremum. In *Optimization Techniques IFIP Technical Conference Novosibirsk, July 1–7, 1974*, pp. 400–404, 1975.

Müller, S., Feure, M., Hollmann, N., and Hutter, F. Pfns4bo: in-context learning for bayesian optimization. In *Proceedings of the 40th International Conference on Machine Learning*, pp. 25444–25470, 2023.

Murphy, K. P. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

Nguyen, X., Wainwright, M. J., and Jordan, M. I. Estimating divergence functionals and the likelihood ratio by convex risk minimization. *IEEE Transactions on Information Theory*, 56(11):5847–5861, nov 2010.

Oliveira, R., Tiao, L. C., and Ramos, F. Batch bayesian optimisation via density-ratio estimation with guarantees. In *Advances in Neural Information Processing Systems*, 2022.

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

Perrone, V., Jenatton, R., Seeger, M. W., and Archambeau, C. Scalable hyperparameter transfer learning. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

Perrone, V., Shen, H., Seeger, M., Archambeau, C., and Jenatton, R. Learning search spaces for bayesian optimization: Another view of hyperparameter transfer learning. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, 2019.

Pineda-Arango, S., Jomaa, H. S., Wistuba, M., and Grabocka, J. HPO-B: A large-scale reproducible benchmark for black-box HPO based on openml. *Neural Information Processing Systems (NeurIPS) Track on Datasets and Benchmarks*, 2021.

Qin, J. Inferences for case-control and semiparametric two-sample density ratio models. *Biometrika*, 85(3):619–630, 1998.

Rasmussen, C. E. *Gaussian Processes in Machine Learning*. The MIT Press, 2004.

Salinas, D., Shen, H., and Perrone, V. A quantile-based approach for hyperparameter transfer learning. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119, pp. 8438–8448. PMLR, 2020.

Salinas, D., Golebiowsk, J., Klein, A., Seeger, M., and Archambeau, C. Optimizing hyperparameters with conformal quantile regression. In *Proceedings of the 40th International Conference on Machine Learning*, 2023.

Saseendran, A., Skubch, K., Falkner, S., and Keuper, M. Shape your space: A gaussian mixture regularization approach to deterministic autoencoders. In *Advances in Neural Information Processing Systems*, volume 34, pp. 7319–7332, 2021.

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

Snoek, J., Larochelle, H., and Adams, R. P. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, volume 25, 2012.

Sobester, A., Forrester, A., and Keane, A. *Engineering design via surrogate modelling: a practical guide*. John Wiley & Sons, 2008.

Song, J., Yu, L., Neiswanger, W., and Ermon, S. A general recipe for likelihood-free Bayesian optimization. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162, pp. 20384–20404, 2022.

Springenberg, J. T., Klein, A., Falkner, S., and Hutter, F. Bayesian optimization with robust bayesian neural networks. In *Advances in Neural Information Processing Systems*, volume 29, 2016.

Srinivas, N., Krause, A., Kakade, S., and Seeger, M. Gaussian process optimization in the bandit setting: No regret and experimental design. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, pp. 1015–1022, 2010.

Sugiyama, M., Suzuki, T., and Kanamori, T. *Density Ratio Estimation in Machine Learning*. Cambridge University Press, 2012.

Swersky, K., Snoek, J., and Adams, R. P. Multi-task bayesian optimization. In Burges, C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K. (eds.), *Advances in Neural Information Processing Systems*, volume 26, 2013.

Tiao, L. C., Klein, A., Seeger, M. W., Bonilla, E. V., Archambeau, C., and Ramos, F. Bore: Bayesian optimization by density-ratio estimation. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139, pp. 10289–10300, 2021.

Tighineanu, P., Skubch, K., Baireuther, P., Reiss, A., Berkenkamp, F., and Vinogradska, J. Transfer learning with gaussian processes for bayesian optimization. In *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*, volume 151, pp. 6152–6181, 2022.

Tighineanu, P., Grossberger, L., Baireuther, P., Skubch, K., Falkner, S., Vinogradska, J., and Berkenkamp, F. Scalable meta-learning with gaussian processes. In *International Conference on Artificial Intelligence and Statistics*, pp. 1981–1989, 2024.

Vanschoren, J. Meta-learning: A survey. *arXiv preprint arXiv:1810.03548*, 2018.

Volpp, M., Fröhlich, L. P., Fischer, K., Doerr, A., Falkner, S., Hutter, F., and Daniel, C. Meta-learning acquisition functions for transfer learning in bayesian optimization. In *International Conference on Learning Representations*, 2020.

Wang, Z. and Jegelka, S. Max-value entropy search for efficient bayesian optimization. In *International Conference on Machine Learning*, pp. 3627–3635, 2017.

Wang, Z., Dahl, G. E., Swersky, K., Lee, C., Mariet, Z., Nado, Z., Gilmer, J., Snoek, J., and Ghahramani, Z. Pre-training helps bayesian optimization too. *arXiv preprint arXiv:2207.03084*, 2022.

Weiss, K., Khoshgoftaar, T. M., and Wang, D. A survey of transfer learning. *Journal of Big data*, 3(1):1–40, 2016.

Wistuba, M. and Grabocka, J. Few-shot bayesian optimization with deep kernel surrogates. In *International Conference on Learning Representations*, 2021.

Wistuba, M., Schilling, N., and Schmidt-Thieme, L. Learning hyperparameter optimization initializations. In *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pp. 1–10, 2015.

Wistuba, M., Schilling, N., and Schmidt-Thieme, L. Scalable gaussian process-based transfer surrogates for hyperparameter optimization. *Mach. Learn.*, 107(1):43–78, 2018.

Yogatama, D. and Mann, G. Efficient Transfer Learning Method for Automatic Hyperparameter Tuning. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*, volume 33, pp. 1077–1085, 2014.

A. Likelihood-free acquisition functions

For completeness, we provide the proofs and derivations for TPE (Bergstra et al., 2011), BORE (Tiao et al., 2021), and LFBO (Song et al., 2022). Recall from Equation (1) that the expected utility function is defined as the expectation of the improvement of the utility function $U(y;\tau)$ over the posterior predictive distribution. Given N observations on the target task in a non-meta-learning setting, for the specific expected improvement (EI) acquisition function, where the utility function is $U(y;\tau) := \max(\tau - y, 0)$, the function reads:

$$\begin{aligned}
\alpha^U(\mathbf{x}; \mathcal{D}_N, \tau) &= \mathbb{E}_{p(y|\mathbf{x},\mathcal{D}_N)}[U(y;\tau)] \\
&= \int_{-\infty}^{\infty} U(y;\tau) p(y \mid \mathbf{x}, \mathcal{D}_N)\, \mathrm{d}y \\
&= \int_{-\infty}^{\tau} (\tau - y) p(y \mid \mathbf{x}, \mathcal{D}_N)\, \mathrm{d}y \\
&= \frac{\int_{-\infty}^{\tau} (\tau - y) p(\mathbf{x} \mid y, \mathcal{D}_N) p(y \mid \mathcal{D}_N)\, \mathrm{d}y}{p(\mathbf{x} \mid \mathcal{D}_N)}\,.
\end{aligned} \tag{11}$$

We follow the prove from Tiao et al. (2021) and consider $\ell(\mathbf{x}) = p(\mathbf{x} \mid y \leq \tau, \mathcal{D}_N)$ and $g(\mathbf{x}) = p(\mathbf{x} \mid y > \tau, \mathcal{D}_N)$. The denominator of the above equation can then be written as:

$$\begin{aligned}
p(\mathbf{x} \mid \mathcal{D}_N) &= \int_{-\infty}^{\infty} p(\mathbf{x} \mid y, \mathcal{D}_N) p(y \mid \mathcal{D}_N)\, \mathrm{d}y \\
&= \ell(\mathbf{x}) \int_{-\infty}^{\tau} p(y \mid \mathcal{D}_N)\, \mathrm{d}y \\
&\quad + g(\mathbf{x}) \int_{\tau}^{\infty} p(y \mid \mathcal{D}_N)\, \mathrm{d}y \\
&= \gamma \ell(\mathbf{x}) + (1 - \gamma) g(\mathbf{x})\,,
\end{aligned} \tag{12}$$

where $\gamma = \Phi(\tau) \coloneqq p(y \leq \tau \mid \mathcal{D}_N)$. The numerator can be evaluated as:

$$\int_{-\infty}^{\tau} (\tau - y) p(\mathbf{x} \mid y, \mathcal{D}_N) p(y \mid \mathcal{D}_N)\, \mathrm{d}y = \ell(\mathbf{x}) \int_{-\infty}^{\tau} (\tau - y) p(y \mid \mathcal{D}_N)\, \mathrm{d}y \tag{13}$$

$$= \ell(\mathbf{x}) \tau \int_{-\infty}^{\tau} p(y \mid \mathcal{D}_N)\, \mathrm{d}y - \ell(\mathbf{x}) \int_{-\infty}^{\tau} y p(y \mid \mathcal{D}_N)\, \mathrm{d}y \tag{14}$$

$$= \gamma \tau \ell(\mathbf{x}) - \ell(\mathbf{x}) \int_{-\infty}^{\tau} y p(y \mid \mathcal{D}_N)\, \mathrm{d}y \tag{15}$$

$$= K \cdot \ell(\mathbf{x})\,, \tag{16}$$

where $K = \gamma\tau - \int_{-\infty}^{\tau} y p(y \mid \mathcal{D}_N)\, \mathrm{d}y$. Therefore the EI acquisition function is equivalent to the γ-relative density ratio up to a constant K,

$$\underbrace{\alpha(\mathbf{x}; \mathcal{D}_N, \tau)}_{\text{expected improvement}} \propto \underbrace{\frac{\ell(\mathbf{x})}{\gamma \ell(\mathbf{x}) + (1 - \gamma) g(\mathbf{x})}}_{\gamma-\text{relative density ratio}} \tag{17}$$

Intuitively, one can think of the configurations $\mathbf{x}$ with $y \leq \tau$ as *good* configurations, and the those with $y > \tau$ as *bad* configurations. Then the density ratio can be interpreted as the ratio between the model's prediction whether the configurations belong to the good or bad class.

The tree-structured Parzen estimator (TPE) (Bergstra et al., 2011) estimates this density ratio by explicitly modeling $\ell(\mathbf{x})$ and $g(\mathbf{x})$ using kernel density estimation for a fixed value of the hyperparameter γ. Within BORE (Tiao et al., 2021), the density ratio is modeled by class probabilities, where $\ell(\mathbf{x}) = p(\mathbf{x} \mid y \leq \tau, \mathcal{D}_N)$ and $g = p(\mathbf{x} \mid y > 0, \mathcal{D}_N)$.

Song et al. (2022) proof that the density ratio acquisition functions are not always equivalent to EI. Bergstra et al. (2011) and Tiao et al. (2021) claim that Equation (13) holds true by assuming $\ell(\mathbf{x})$ is independent of y once $y \leq \tau$ and therefore can be treated as a constant inside the integral. In fact, $p(\mathbf{x} \mid y \leq \tau, \mathcal{D}_N)$ still depends on y even if $y \leq \tau$, because it is a

conditional probability conditioned on y not $y \leq \tau$. Therefore, satisfying the condition $y \leq \tau$ does not imply independence of y. From the definition of conditional probability

$$p(\mathbf{x} \mid y \leq \tau, \mathcal{D}_N) = \frac{\int_{-\infty}^{\tau} p(\mathbf{x}, y \mid \mathcal{D}_N)\,\mathrm{d}y}{\int_{-\infty}^{\tau} p(y \mid \mathcal{D}_N)\,\mathrm{d}y} \neq p(\mathbf{x} \mid y, \mathcal{D}_N)\,, \tag{18}$$

we can see that they are not equivalent. Intuitively, the probability of the configuration $\mathbf{x}$ for a given y value should still depend on y even if $y < \tau$ holds. By making this independence assumption, the resulting density ratio acquisition function treats all $(\mathbf{x}, y)$ pairs below the threshold with equal probability (importance), when, in fact, EI weights the importance of $(\mathbf{x}, y)$ pairs by the utility $\max(\tau - y, 0)$

To tackle this issue, Song et al. (2022) propose to directly approximate EI inspired by the idea of variational f-divergence estimation (Nguyen et al., 2010). They provide a variational representation for the expected utility function at any point $\mathbf{x}$, provided samples from $p(y \mid \mathbf{x})$. Thereby, their approach replaces the potentially intractable integration with the variational objective function that can be solved based on samples:

$$\mathbb{E}_{p(y|\mathbf{x})}[U(y;\tau)] = \underset{s\in[0,\infty)}{\arg\max}\, \mathbb{E}_{p(y|\mathbf{x})}[U(y;\tau)f'(s)] - f^\star(f'(s))\,, \tag{19}$$

where the utility function $U : \mathbb{R} \times \mathcal{T} \to [0, \infty)$ is non-negative, $\tau \in \mathcal{T}$, $f : [0, \infty) \to \mathbb{R}$ is a strictly convex function with third order derivatives, and $f^\star$ is the convex conjugate of f. The maximization is performed over $s \in [0, \infty)$ and it does not model distributions with probability but only samples from the observations $\mathcal{D}_N$.

They consider the approximated expected utility acquisition function as $\alpha^{\text{LFBO}} = \hat{S}_{\mathcal{D}_{N,\tau}}(\mathbf{x})$, which can be written as:

$$\hat{S}_{\mathcal{D}_{N,\tau}}(\mathbf{x}) = \underset{S:\mathcal{X}\to\mathbb{R}}{\arg\max}\, \mathbb{E}_{\mathcal{D}_N}[U(y;\tau)f'(S(\mathbf{x})) - f^\star(f'(S(\mathbf{x}))]. \tag{20}$$

By optimizing a variational objective in the search space $\mathcal{X}$, the expected utility acquisition function over $\mathbf{x}$ can be recovered. For practical purpose, they choose a specific convex function f: $f(r) = r \log \frac{r}{r+1} + \log \frac{1}{r+1}$ for all $r > 0$, and a specific form of $S = C/(1 - C)$, where $C : \mathcal{X} \to (0, 1)$ and can be considered as a probabilistic classifier. By applying these into Equation (20), the resulting acquisition function reads:

$$\alpha^{\text{LFBO}}(\mathbf{x}; \mathcal{D}_N, \tau) = \hat{S}_{\mathcal{D}_N,\tau}(\mathbf{x}) = \hat{C}_{\mathcal{D}_N,\tau}(\mathbf{x})/(1 - \hat{C}_{\mathcal{D}_N,\tau}(\mathbf{x})), \tag{21}$$

where $\hat{C}_{\mathcal{D}_N,\tau}$ is the maximizer of an objective over C:

$$\mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}_N}[U(y;\tau)\ln C(\mathbf{x}) + \ln(1 - C(\mathbf{x}))]\,. \tag{22}$$

This is can be reinterpreted as a classification loss with training examples weighted by the utility function.

B. Probit approximation

While one can sample the predictive posterior to make class prediction as in Equation (9), an alternative way is to approximate the integral in Equation (8) via probit approximation. Let $a = m(\mathbf{\Phi}) + \mathbf{z}^\mathsf{T}\mathbf{\Phi}$ and $q(\mathbf{z}) = \mathcal{N}(\mathbf{z} \mid \mathbf{z}_{\text{MAP}}, \mathbf{\Sigma}_N)$ be the approximated posterior obtained through the Laplace approximation. The distribution of a then follows the Gaussian $\mathcal{N}(a \mid \mu_a, \sigma_a^2)$ with the following parameters:

$$\begin{aligned} \mu_a = \mathbb{E}[a] &= \int p(a)a\,da \\ &= \int q(\mathbf{z})(m(\mathbf{\Phi}) + \mathbf{z}^\mathsf{T}\mathbf{\Phi})\,d\mathbf{z} \\ &= m(\mathbf{\Phi}) + \mathbf{z}_{\text{MAP}}^\mathsf{T}\mathbf{\Phi}\,, \end{aligned} \tag{23}$$

$$\begin{aligned} \sigma_a^2 &= \int p(a)[a^2 - \mathbb{E}[a]^2]\,\mathrm{d}a \\ &= \int q(\mathbf{z})\left((m(\mathbf{\Phi}) + \mathbf{z}^\mathsf{T}\mathbf{\Phi})^2 - (m(\mathbf{\Phi}) + \mathbf{z}_{\text{MAP}}^\mathsf{T}\mathbf{\Phi})^2\right)\,\mathrm{d}\mathbf{z} \\ &= \mathbf{\Phi}^\mathsf{T}\mathbf{\Sigma}_N\mathbf{\Phi}\,. \end{aligned} \tag{24}$$

Thus our approximation to the predictive distribution in Equation (8) becomes

$$C(\mathbf{x}) \simeq \int p(k=1 \mid \boldsymbol{\omega}, \mathbf{z}) q(\mathbf{z}) \, \mathrm{d}\mathbf{z} = \int \sigma(a) \mathcal{N}(a \mid \mu_a, \sigma_a^2) \, \mathrm{d}a \,. \tag{25}$$

Since the integral in Equation (25) cannot be evaluated analytically due to the sigmoid function, we need to approximate it to obtain the marginal class prediction. One can approximate the integral by exploiting the similarity between the logistic sigmoid function $\sigma(a)$ and the inverse probit function Bishop & Nasrabadi (2006); Murphy (2012), which is given by the cumulative distribution of the standard Gaussian $\Phi(a)$. In order to obtain good approximation results, we need to rescale the horizontal axis so that $\sigma(a)$ has the same slope as $\Phi(\lambda a)$, where $\lambda^2 = \pi/8$. By replacing $\sigma(a)$ with $\Phi(\lambda a)$ in Equation (25), we obtain the approximated predictive distribution:

$$\begin{aligned} p(k=1 \mid \boldsymbol{\omega}, \mathcal{D}_N) &\approx \int \Phi(\lambda a) \mathcal{N}(a \mid \mu_a, \sigma_a^2) \, \mathrm{d}a \\ &= \Phi\left(\frac{\mu_a}{(\lambda^{-2} + \sigma_a^2)^{1/2}} \right) \\ &= \sigma\left((1 + \pi \sigma_a^2 / 8)^{-1/2} \mu_a \right) . \end{aligned} \tag{26}$$

C. Regularizing the latent task space

To make sure our task distribution conforms to the prior distribution $p(\mathcal{Z})$, we followed the approach in Saseendran et al. (2021), where they regularize the learned latent representation towards a given prior distribution in a tractable way. Their approach builds on the non-parametric Kolmogorov-Smirnov (KS) test for one-dimension probability distributions and extend it to a multivariate setting, which allows for gradient-based optimization and can be easily applied to expressive multi-modal prior distributions.

Directly extending the KS test to high-dimensional distributions is challenging, since it requires matching joint CDFs, which is especially infeasible in this case. Therefore, they propose to match the marginal CDFs of the prior, making the regularization tractable. Given d-dimensional task embedding $\mathbf{z}_1, \ldots, \mathbf{z}_T$ for T related tasks, the empirical CDF in dimension j is defined as:

$$F(z) = \frac{1}{T} \sum_{t=1}^{T} \mathbb{1}([\mathbf{z}_t]_j \leq z) \,, \tag{27}$$

where $\mathbb{1}([\mathbf{z}_t]_j \leq z)$ is a indicator function if j-th component of $\mathbf{z}_t$ is smaller or equal than a certain value z. In addition to the marginals, they also regularize the empirical covariance matrix $\mathrm{Cov}(\mathbf{z}_1, \ldots, \mathbf{z}_T)$ to be close to the covariance of the prior. In our case, the prior task distribution is a isotropic Gaussian $p(\mathcal{Z}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$, and the resulting regularizer can be written as:

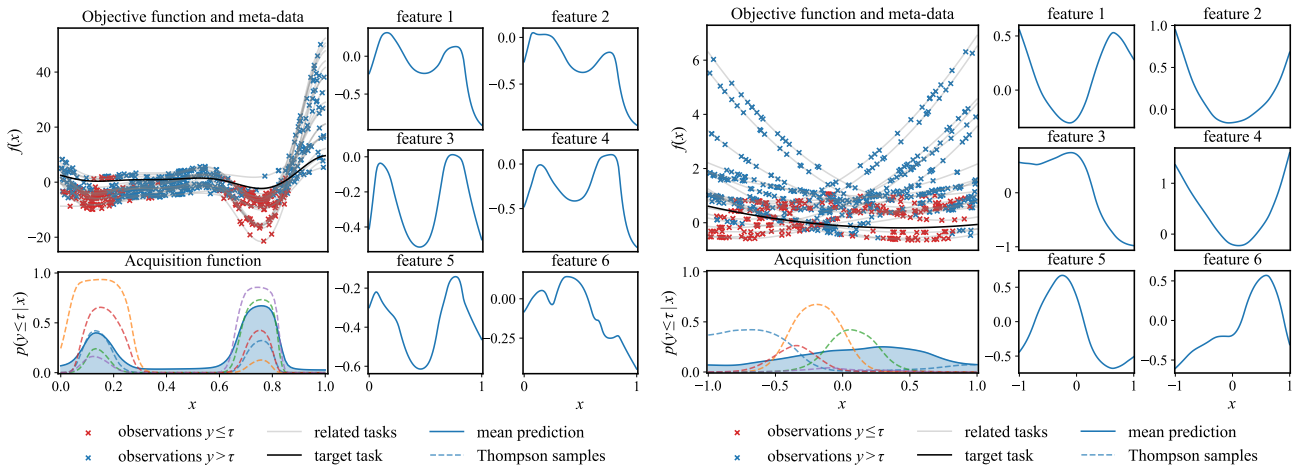
$$\mathcal{R}(\{\mathbf{z}_t\}_{t=1}^T; p(\mathcal{Z})) = \lambda_{\mathrm{KS}} \underbrace{\sum_{j=1}^{d} (F([\mathbf{z}_t]_j) - \Phi([\mathbf{z}_t]_j))^2}_{\text{match marginal CDF of } p(\mathcal{Z})} + \underbrace{\lambda_{\mathrm{Cov}} \|\mathbf{I} - \mathrm{Cov}(\{\mathbf{z}_t\}_{t=1}^T)\|_{\mathrm{F}}^2}_{\text{match covariance of } p(\mathcal{Z})} \,, \tag{28}$$

where the marginal CDFs and correlations are compared through squared errors, the λ_{KS} and λ_{CV} are weighting factors controls the trade-off between matching the empirical marginal CDFs and the covariance.

Regularization coefficients estimation The two regularization coefficients λ_{KS} and λ_{Cov} are important hyperparameters and needed to be carefully treated. Notice that, the more tasks we have, the closer the match between the empirical CDFs and the prior marginal CDF based on the assumption that the related tasks are i.i.d. samples from the task distribution. To avoid the regularization being dominated by one term, we aim to scale them in a way that all terms converge to similar magnitudes with more tasks. Therefore, we choose the two factors such that

$$\begin{aligned} \lambda_{\mathrm{KS}}^{-1} &= 2 \sum_{j=1}^{d} (F([\mathbf{z}_t]_j) - \Phi([\mathbf{z}_t]_j))^2 \,, \quad \text{with } \mathbf{z}_t \sim p(\mathcal{Z}) \,, \\ \lambda_{\mathrm{Cov}}^{-1} &= 2 \|\mathbf{I} - \mathrm{Cov}(\{\mathbf{z}\}_{t=1}^T)\|_{\mathrm{F}}^2 \,, \quad \text{with } \mathbf{z}_t \sim p(\mathcal{Z}) \,, \end{aligned} \tag{29}$$

where $\mathbf{z}_t$ is i.i.d. sample from the prior distribution for the coefficient estimation. This normalizes the regularizer to be approximately of order 1 for samples follows the prior distribution.



(a) Forrester function features

(b) Quadratic function features

Figure 8. Left: Forrester functions with two likely optima as target function and related tasks. The learned acquisition function is shown below. The meta-learned latent features show that the model successfully infers the location of two optima, resulting in a acquisition with two modes around the optima. Right: Quadratic functions with varying optima as target function and related tasks. The meta-learned latent features show that the model is able to capture the global function shape shared across all tasks, even though there is no clear location for optima.

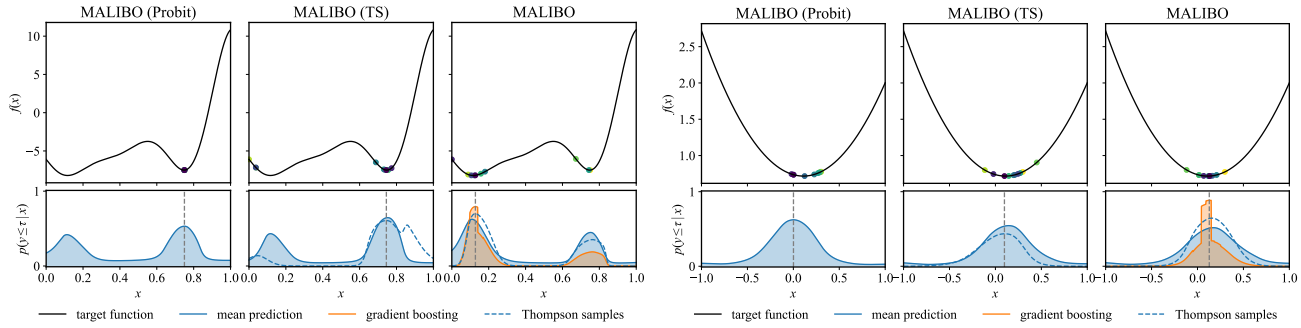
D. Ablation studies

In this section, we conduct ablation studies to illustrate the impact of different components of MALIBO on its performance. We introduce the following variants of MALIBO as in Section 5:

- MALIBO (Probit): Utilizes the marginalized form of the acquisition function (see Appendix B) without gradient boosting.
- MALIBO (TS): Employs only Thompson sampling without gradient boosting.
- MALIBO (RES): Removes the mean prediction layer $m(\cdot)$ while keeping other components unchanged.
- MALIBO (MEAN): Excludes the task prediction layer $h_{\mathbf{z}_t}(\cdot)$ and uses only the task-agnostic meta-learning component $g_{\boldsymbol{\omega}}$ with gradient boosting. This variant focuses on meta-learning the initial design for optimization. Note that, due to the absence of task embedding, the model is trained without the task space regularization in Equation (4), making the Thompson sampling strategy inapplicable.
- MALIBO (RF): Replaces gradient boosting with a random forest (RF) classifier, using the implementation from scikit-learn (Pedregosa et al., 2011). Following Song et al. (2022), the hyperparameters are set as: n_estimator $= 1000$, min_samples_split $= 2$, max_depth $=$ None, min_samples_leaf $= 1$. Unlike in gradient boosting, this variant requires explicit balancing of the results from the meta-learning and residual models, which we achieve by averaging their predictions without applying a sophisticated weighting scheme.
- MALIBO (MLP): Substitutes gradient boosting with a two-layer multi-layer perceptron (MLP) classifier, with 32 hidden units per layer and ReLU activation. The MLP is optimized using ADAM (Kingma & Ba, 2015) with learning rate lr $= 10^{-3}$ and batch size $B = 64$. Predictions from the meta-learning and residual models are averaged similarly to the RF variant.

D.1. Latent feature analysis

To provide an intuition of the meta-learning in our method, we visualize the feature representation extracted from the meta-data by our meta-learning model. The latent features $\boldsymbol{\Phi}$ represent basis functions for the Bayesian logistic regression and should represent the structure of the meta-data distribution. With successfully learned features $\boldsymbol{\Phi}$ and the mean layer,



(a) Forrester function features

(b) Quadratic function features

Figure 9. Task adaptation of different MALIBO variants on Forrester and quadratic functions after meta-learning. Each method optimizes for 10 iterations.

our model performs the task adaptation by reasoning about the latent task embedding vector $\mathbf{z}$. It produces predictions with similar structure to the meta-data that match the class labels on the target function. In order to learn a effective feature representation, one should capture both the local and global structure of the function. Therefore, we select two types of function to study the effectiveness of feature learning for MALIBO: i) Forrester functions (Sobester et al., 2008) with two very likely positions for the global optimum, which allows for effective warm-starting and requires local adaptation. ii) quadratic functions, where the functions share a certain global shape, but the optima could be located anywhere in the search space. For more details on the synthetic functions and the generation of meta-data, we refer to Appendix H.

The results for these two synthetic functions are shown in Figure 8a and Figure 8b respectively. In Figure 8a, we observe that the features learned by MALIBO exhibit either a maximum or a minimum around the two likely optima, indicating that the model successfully infers the location of the most promising values from the meta-data. In Figure 8b, even without a clear location of optima, the features still follow the shape of quadratic functions with different minima.

D.2. Effects of Thompson Sampling

To understand how the exploration help with the optimization, we compare the task adaptation performance among MALIBO (Probit), MALIBO (TS) and MALIBO on synthetic benchmarks. As illustrated in Figure 9, MALIBO (Probit) tends to exhibit conservative behavior in both the Forrester and quadratic function scenarios, leading to sub-optimal performance. This is primarily due to its limited exploration capabilities. In contrast, MALIBO (TS), which incorporates Thompson sampling, demonstrates more robust exploration in both cases. Interestingly, even though MALIBO is enhanced with gradient boosting, its exploration performance appears similar to the Thompson sampling-only variant. The influence of gradient boosting is evident in the change of acquisition function values. It suppresses the values in regions where the sampled function might predict high values, but existing observations suggest otherwise. Conversely, it amplifies the acquisition function values near the current best observations, thereby fostering stronger convergence. In the initial stages of optimization, both task adaptation and gradient boosting face a challenge due to the scarcity of observations, which limits confident prediction. Unlike other methods that rely on random search as an exploration strategy, Thompson sampling incorporates task uncertainty. It efficiently explores potential optima by leveraging meta-learned information from related tasks, making it a more effective approach in the context of exploration and optimization.

D.3. Effects of gradient boosting

To illustrate the effectiveness of gradient boosting in MALIBO, we conduct an experiment that focus on this aspect by excluding meta-learning. This approach mimics scenarios where meta-learning fails to aid task adaptation. As shown in Figure 10, our experiments compare various MALIBO variants on a Forrester function, and contrast these results with LFBO to assess their performances without meta-learning. The findings, depicted in Figure Figure 10, reveal that MALIBO (Probit) and MALIBO (TS) are inefficient in optimizing the function due to their exclusive reliance on meta-learned features for task adaptation. This reliance results in poor performance when the meta-learned priors are uninformative. In contrast, the proposed MALIBO efficiently locates the optima and performs similarly to LFBO. This efficiency is attributed to the ability of gradient boosting to counteract ineffective predictions from the meta-learning process. Specifically, gradient

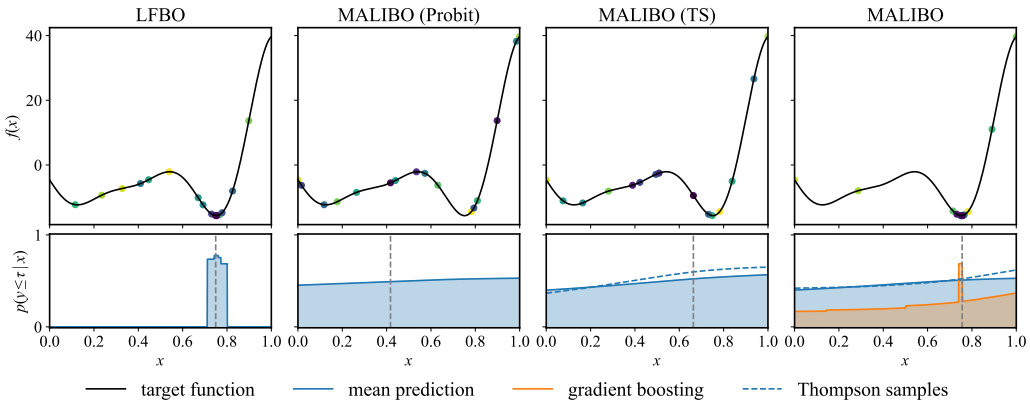


Figure 10. Task adaptation of LFBO and different MALIBO variants on a Forrester function without meta-learning. Each method optimize for 16 iterations

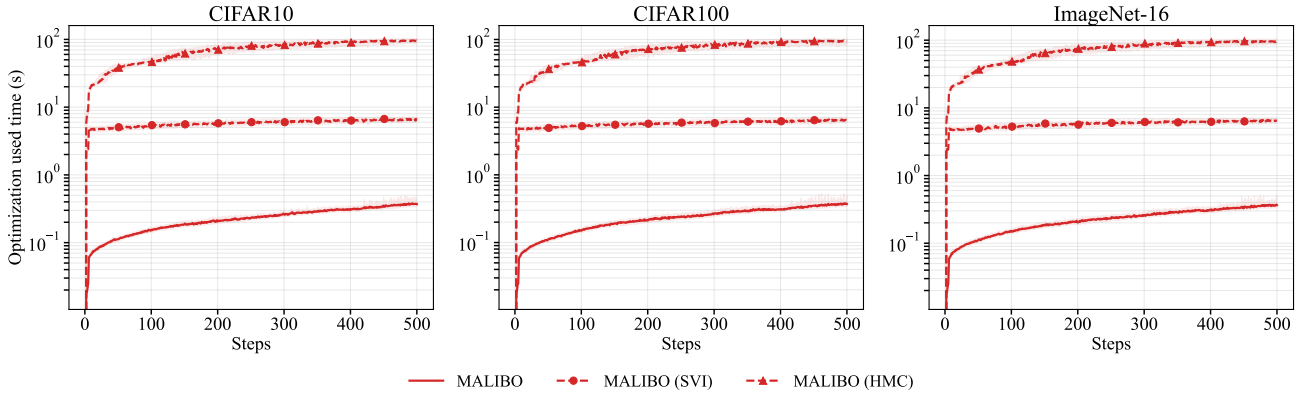


Figure 11. Runtime of MALIBO using different inference methods over optimization steps on NASBench201. We plot the medial inter-quantiles to remove outliers.

boosting enables subsequent learners to correct initial errors from the meta-learned model, thereby aligning the performance of MALIBO with that of LFBO.

D.4. Effects of different inference methods

In this section, we investigate the performance of different approximation methods for the posterior task embedding $p(\mathbf{z} \mid \mathcal{D}_N)$. Specifically, we consider three different inference methods, namely Hamiltonian Monte Carlo (HMC), stochastic variational inference (SVI) and the Laplace approximation. Compared to the Laplace approximation, SVI and HMC normally take longer time for the approximation, especially for HMC, as it needs multiple samples to estimate the expectation. To show their speed and scalability, we compare their runtime for optimization on NASBench201 in Figure 11 and show that the MALIBO with Laplace approximation takes around 0.1 second for every iteration, while MALIBO (SVI) takes around 10 seconds and MALIBO (HMC) 100 seconds. Although the SVI and HMC variants take longer time for inference, we show that the performance among these methods are close in Figure 12. Similar behaviors are also observed in other benchmarks, including the HPOBench and HPO-B. Due to the fast inference time and competitive performance, we use Laplace approximation as our proposed inference method for MALIBO.

D.5. Effects of task embedding dimension

Given that the task embedding dimension $d = 50$ is fixed across all experiments, its impact on the performance of MALIBO remains uncertain. To address this, we conduct an ablation study using the HPO-B benchmark, which encompasses tasks with input dimensions ranging from 2 to 18. We test various task embedding dimensions $d = 4, 8, 16, 32, 50, 64, 128$. As

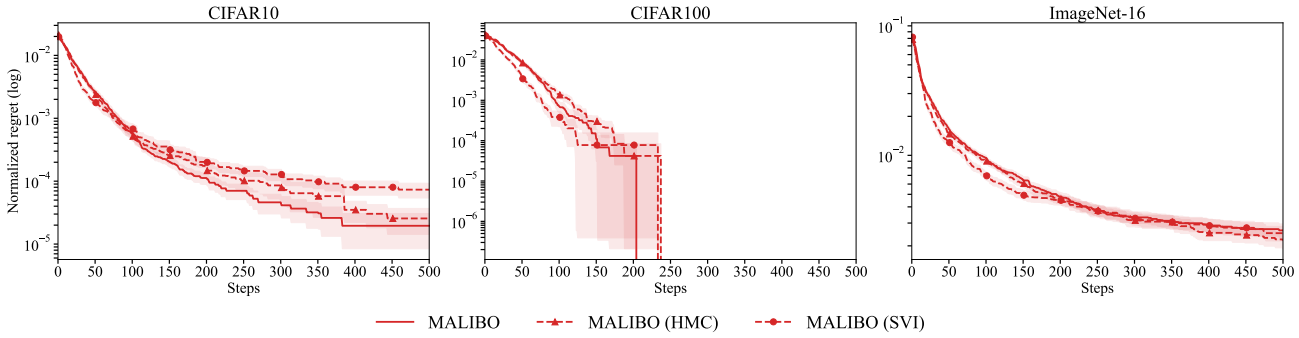


Figure 12. Normalized regrets of MALIBO using different inference methods on NASBench201.

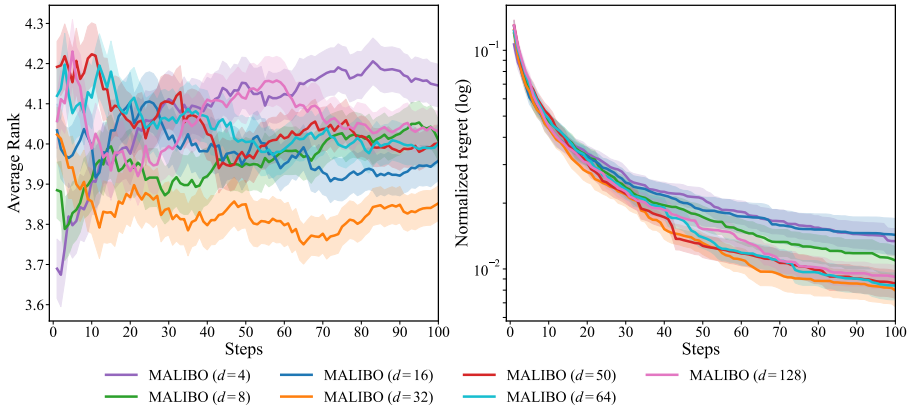


Figure 13. Aggregated comparisons across all search spaces for MALIBO with different task embedding dimensions on HPO-B.

illustrated in Figure 13, lower dimensional embeddings tend to yield better initial performance. However, this performance often plateaus, possibly due to the embeddings' limited expressiveness. Conversely, while increasing the dimensionality generally enhances performance, this improvement persists only up to a certain point, specifically $d = 32$ in our study, beyond which the influence of task embedding dimensionality on performance diminishes.

D.6. Quantitative comparison

In this section, we demonstrate the detailed experimental results for the quantitative ablation study that is introduced in Section 5. This study, illustrated in Figures 14 to 16, compares seven variants of MALIBO across all real-world benchmarks. These variants include MALIBO (Probit), MALIBO (TS), MALIBO (MEAN), MALIBO (RES), MALIBO (RF), MALIBO (MLP) and the proposed MALIBO. We evaluated the performance using immediate regrets, which is the absolute error between the global minimum and the best evaluated results so far. This metric was applied to both HPOBench and NASBench201. For HPO-B, we followed the plotting protocol established by Pineda-Arango et al. (2021)., where the plots include the normalized regrets and average rank across benchmarks, alongside with the critical difference diagram (Demšar, 2006) for the ranks of all runs @25, @50, and @100 steps to assess the statistical significance of methods.

E. Additional results

In this section, we present results related to the benchmarks discussed in Section 5. To further examine the robustness against heteroscedastic noise, we demonstrate additional experiments on two more synthetic functions. For the real-world benchmark, we include comprehensive results for all target tasks within the benchmarks, offering a more detailed analysis. Additionally, we provide a runtime analysis to demonstrate the the scalability of our proposed method.

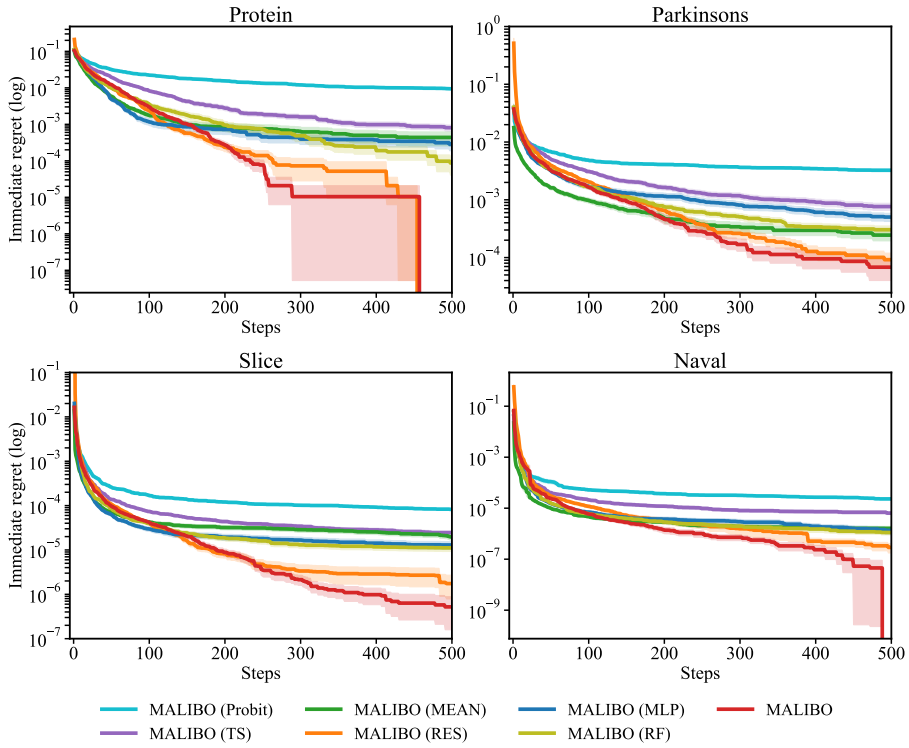


Figure 14. Immediate regrets of MALIBO variants on all tasks in HPOBench.

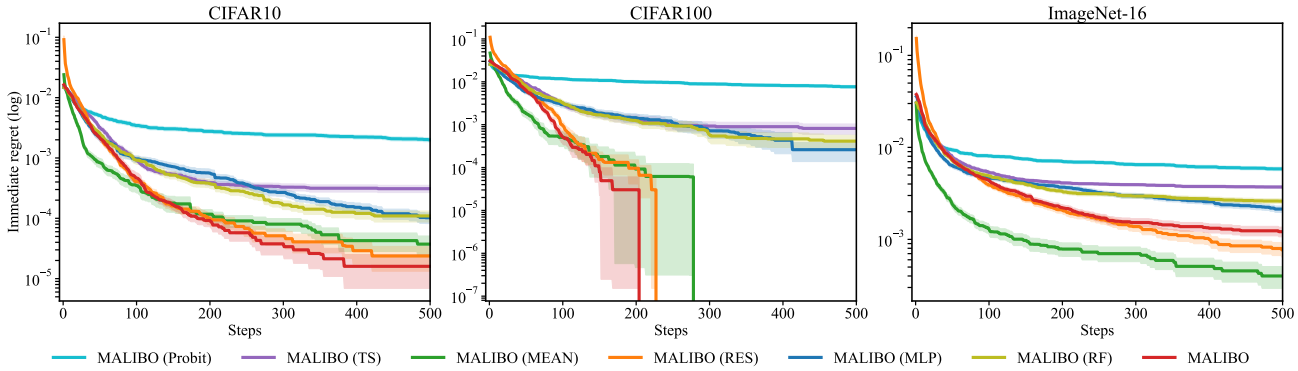


Figure 15. Immediate regrets of MALIBO variants on all tasks in NASBench201.

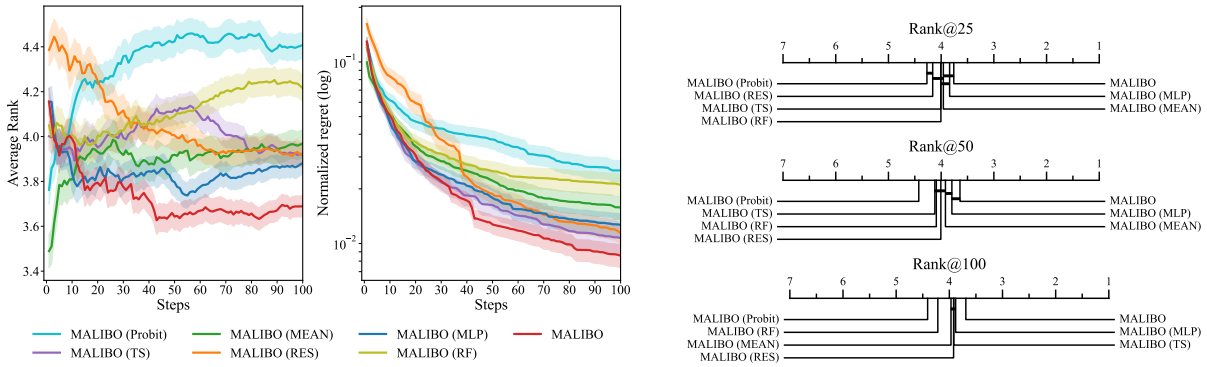


Figure 16. Aggregated comparisons of normalized regret and average ranks across all search spaces for MALIBO variants on HPO-B.

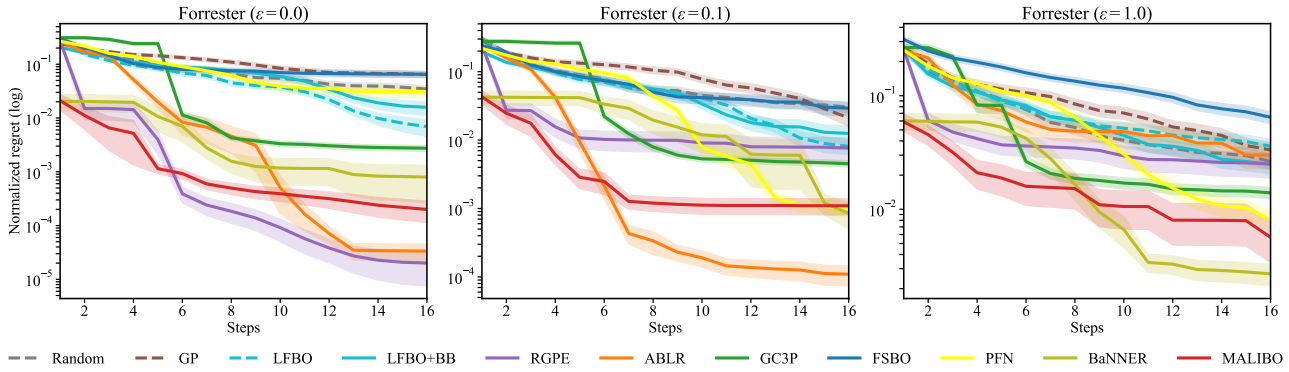


Figure 17. Normalized regret for BO algorithms on Forrester function ensembles ($D = 1$) with different levels of multiplicative noise.

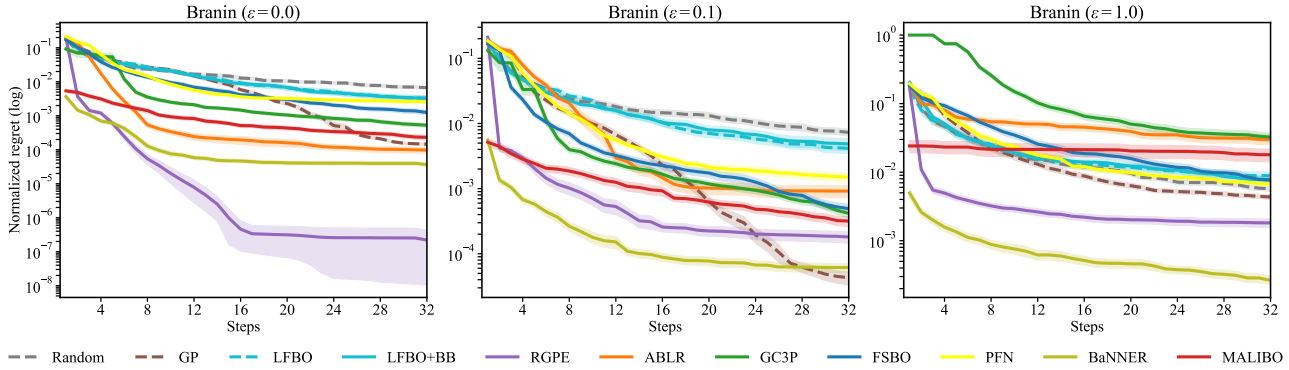


Figure 18. Normalized regret for BO algorithms on Branin function ensembles ($D = 2$) with different levels of multiplicative noise.

E.1. Noise experiment

In this experiments, we use the Forrester(Sobester et al., 2008) and Branin (Dixon, 1978) function ensembles as additional benchmarks, with detailed descriptions available in Appendix F. For meta-learning, we randomly sampled N noisy observations in T related tasks, setting $N = 128, T = 128$ for Forrester and $N = 128, T = 256$ for Branin. As illustrated in Figures 17 and 18, MALIBO consistently demonstrate strong warm-starting performance and stay robust to noise compared to most of the baselines. Similarly, the performances of LFBO and LFBO+BB are relatively stable across different noise levels but are only comparable to random search. Notably, while RGPE and ABLR both outperform the other likelihood-free based methods in the noise-free setting, their performances degrade significantly with increased noise levels except for RGPE in Branin ($\epsilon = 1.0$).

E.2. Runtime analysis

The runtime efficiency of MALIBO is investigated in Section 5, where we illustrate the runtime of the optimization algorithms for each step in Figure 4. The runtime for MALIBO is the second fastest among all the meta-learning methods, while only slightly slower than LFBO and LFBO+BB. Due to the increasing amount of observations, the runtime of almost all the methods grows over with the number of iterations, especially for RGPE and PFN. The most time-consuming methods are FSBO, BaNNER and PFN, which take almost 100 seconds for each steps. For PFN and BaNNER, this is mostly from optimizing the acquisition function, which requires multiple initializations to guarantee better convergence to the global optimum. In terms of FSBO, the time overhead also comes from the additional training in task-adaptation phase besides the acquisition function optimization. Although ABLR and GC3P are around one order of magnitude slower than MALIBO at the beginning, but their runtime remain stable throughout the optimization. ABLR always retrain on all data, which constitutes the largest computational burden at each step, and the complexity of the Bayesian linear regression is more scalable than GPs. For GC3P, we attribute the almost constant runtime to aggressive settings for the GPs hyperparameter optimization, which is usually the most expensive step. The growth in runtime for LFBO and LFBO+BB can be attributed

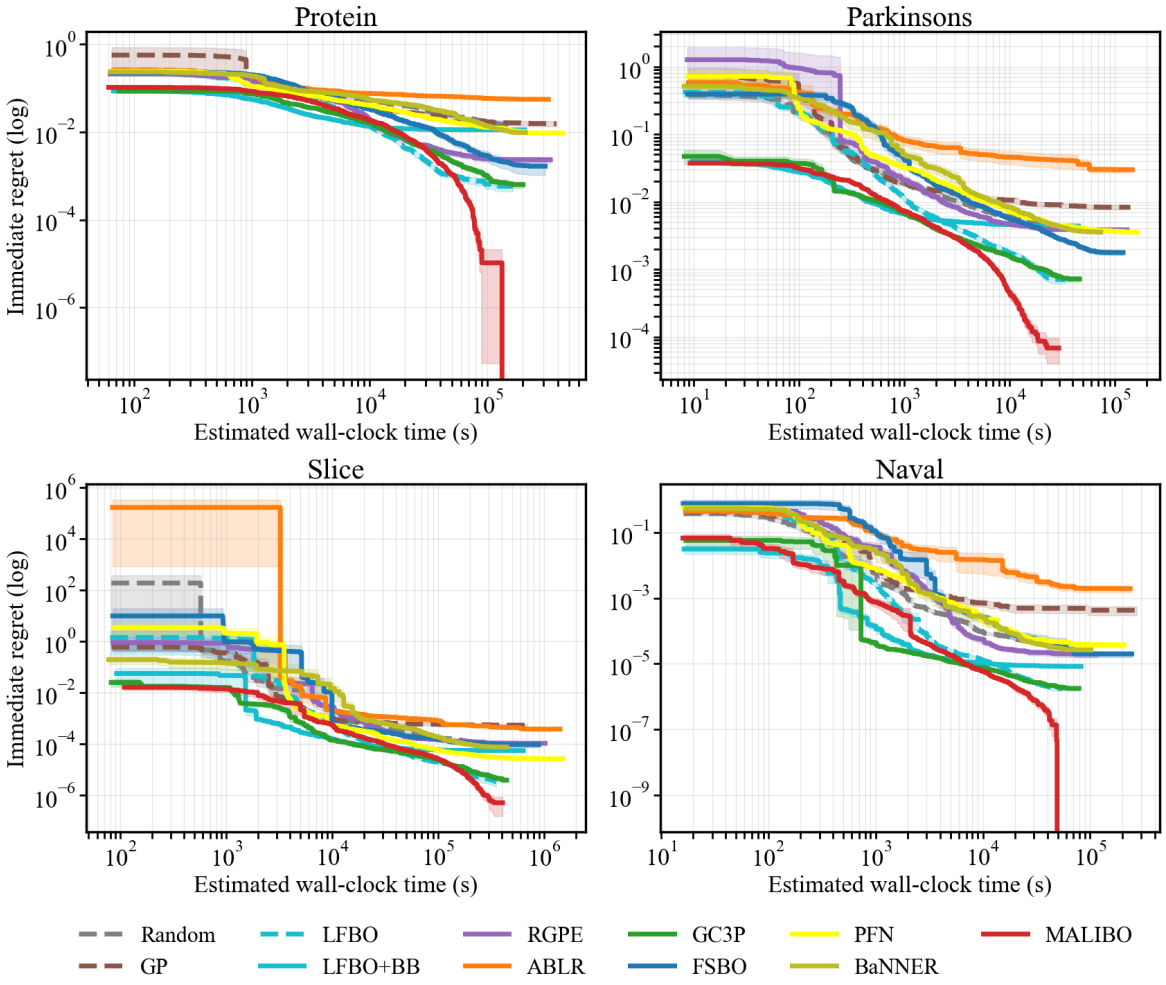


Figure 19. Immediate regrets of different BO algorithms on the HPOBench neural network tuning problem. Each algorithm runs for 500 iterations and we show the corresponding estimated wall-clock time on the x axis in log scale.

exclusively to the fitting of the gradient boosted trees. Similarly, MALIBO uses gradient boosting as residual prediction model, which retrains on the dataset for every iteration, therefore the runtime grows with the number of iterations as well.

In addition to the runtime, we also report results for HPOBench and NASBench201 focusing on immediate regrets as a function of the estimated wall-clock time[2]. To obtain the realistic wall-clock time, we accumulate the time to optimize for corresponding BO methods and the recorded runtime for the configurations in the benchmarks. Notice that all the methods run for the same number of steps in an experiment. The results in Figures 19 and 20 show that MALIBO attains the best warm-starting performance across almost all benchmarks and constantly achieves one of the lowest final regrets in the same amount of time.

E.3. Real-world benchmarks

In Figures 21 to 23, we demonstrate the complementary results for the real-world benchmarks that is introduced in Section 5. Additionally, we report the results for two more recent baselines, namely OptFormer (Chen et al., 2022) and NAP (Maraval et al., 2023) in Figures 26 to 28. However, due to the excessive training time of these two methods, we only compare our baselines in 6 representative search spaces in HPO-B benchmark as in (Maraval et al., 2023).

[2]We have limited the runtime analysis to only HPOBench and NASBench201 because HPO-B lacks the necessary runtime information for such an evaluation.

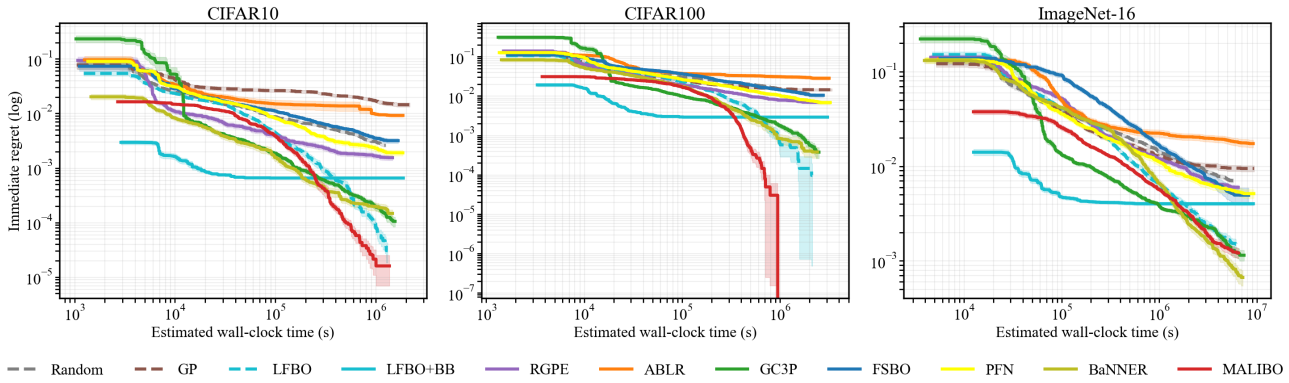


Figure 20. Immediate regrets of different BO algorithms on the NASBench201 neural network architecture search problem. Each algorithm runs for 500 iterations and we show the corresponding estimated wall-clock time on the x axis in log scale.

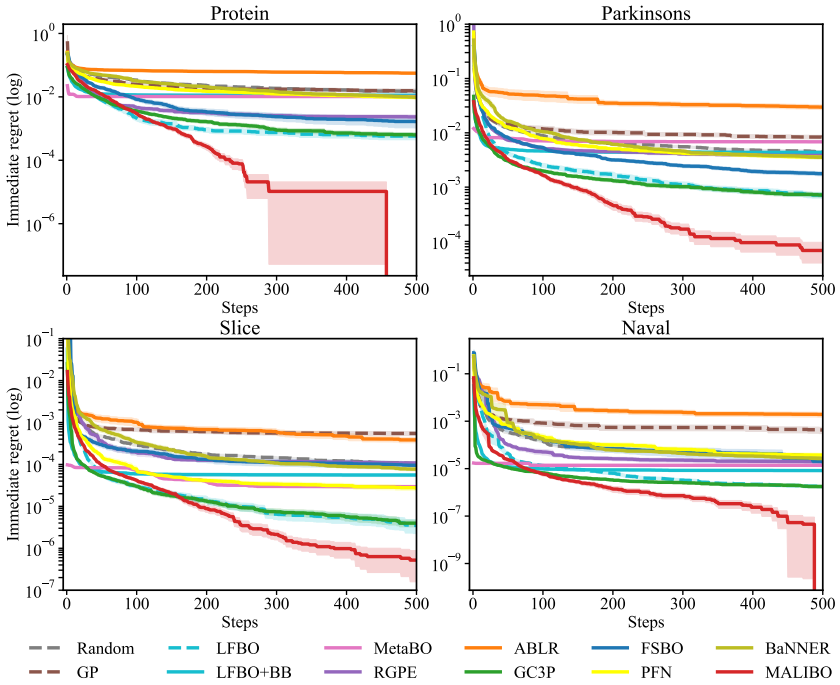


Figure 21. Immediate regrets for BO algorithms on HPOBench for 4 datasets.

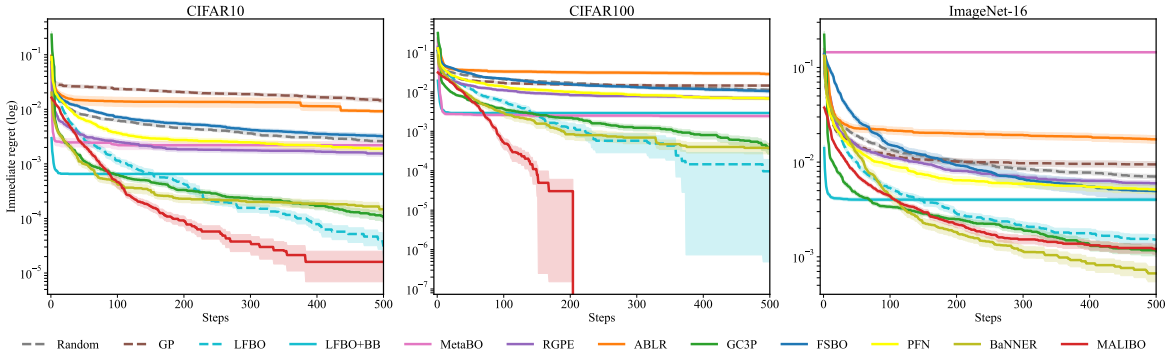


Figure 22. Immediate regrets for different BO algorithms on NASBench201 for 3 datasets.

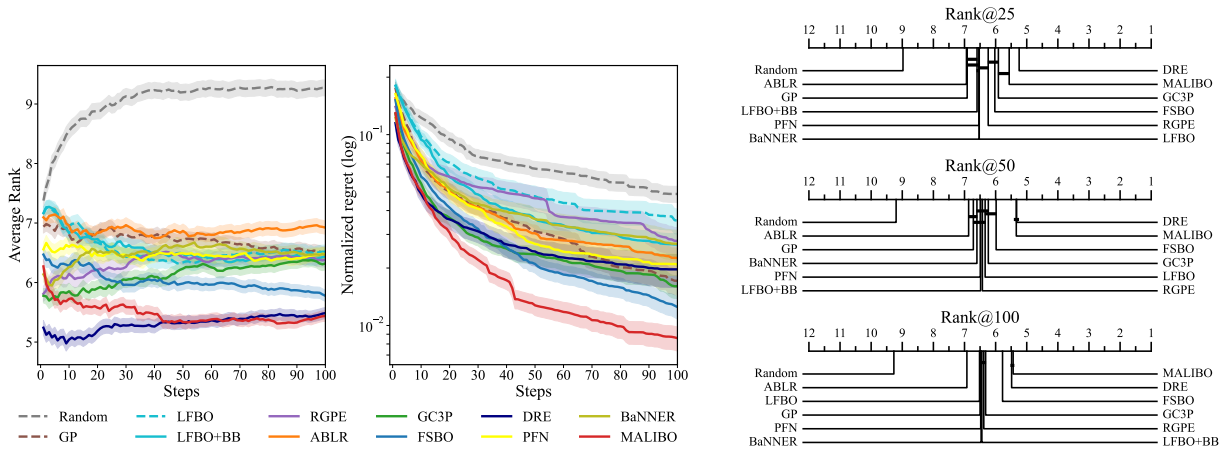


Figure 23. Aggregated comparisons of normalized regret and average ranks across all search spaces for BO methods on HPO-B.

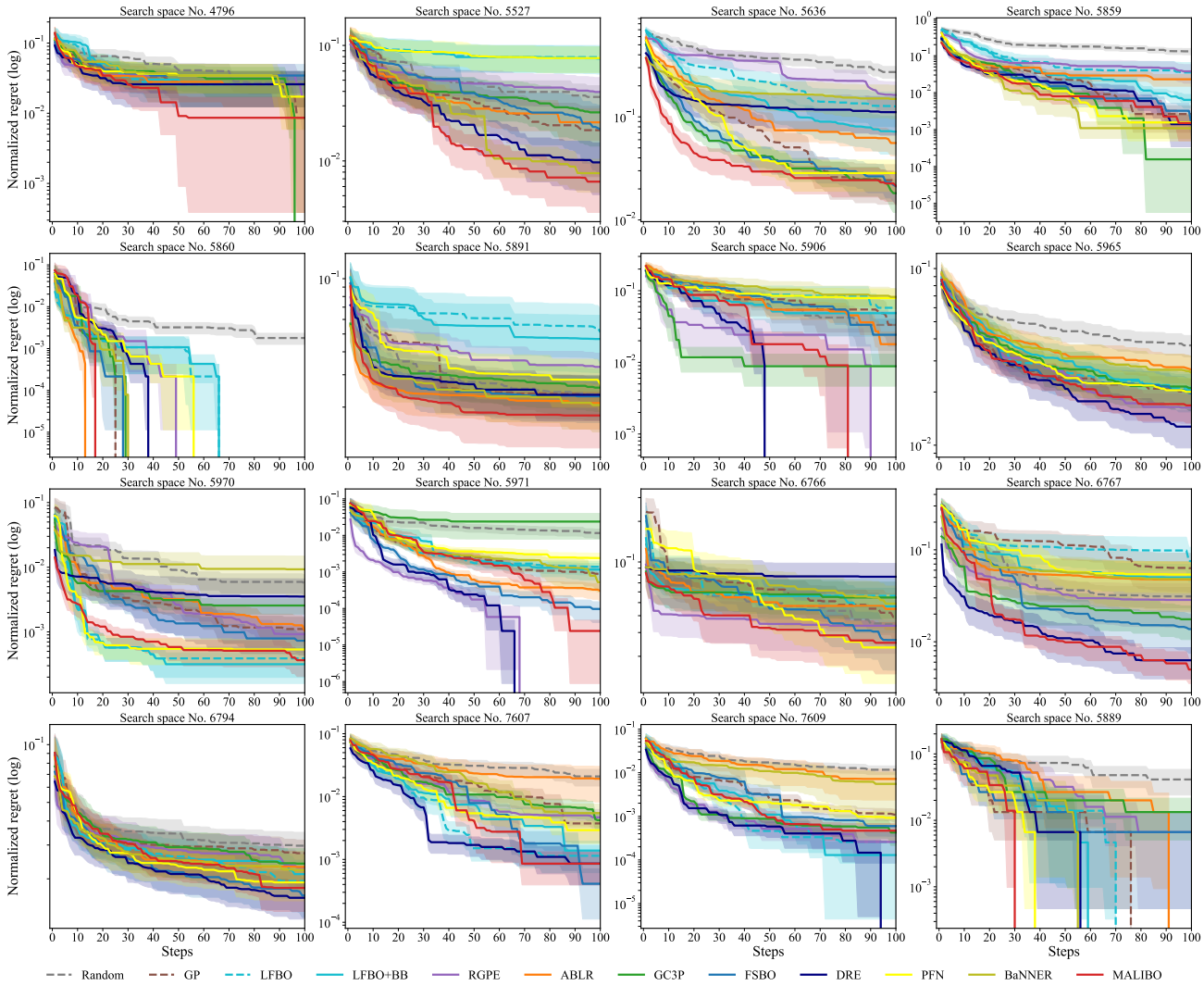


Figure 24. Normalized regret comparison of BO methods on HPO-B.

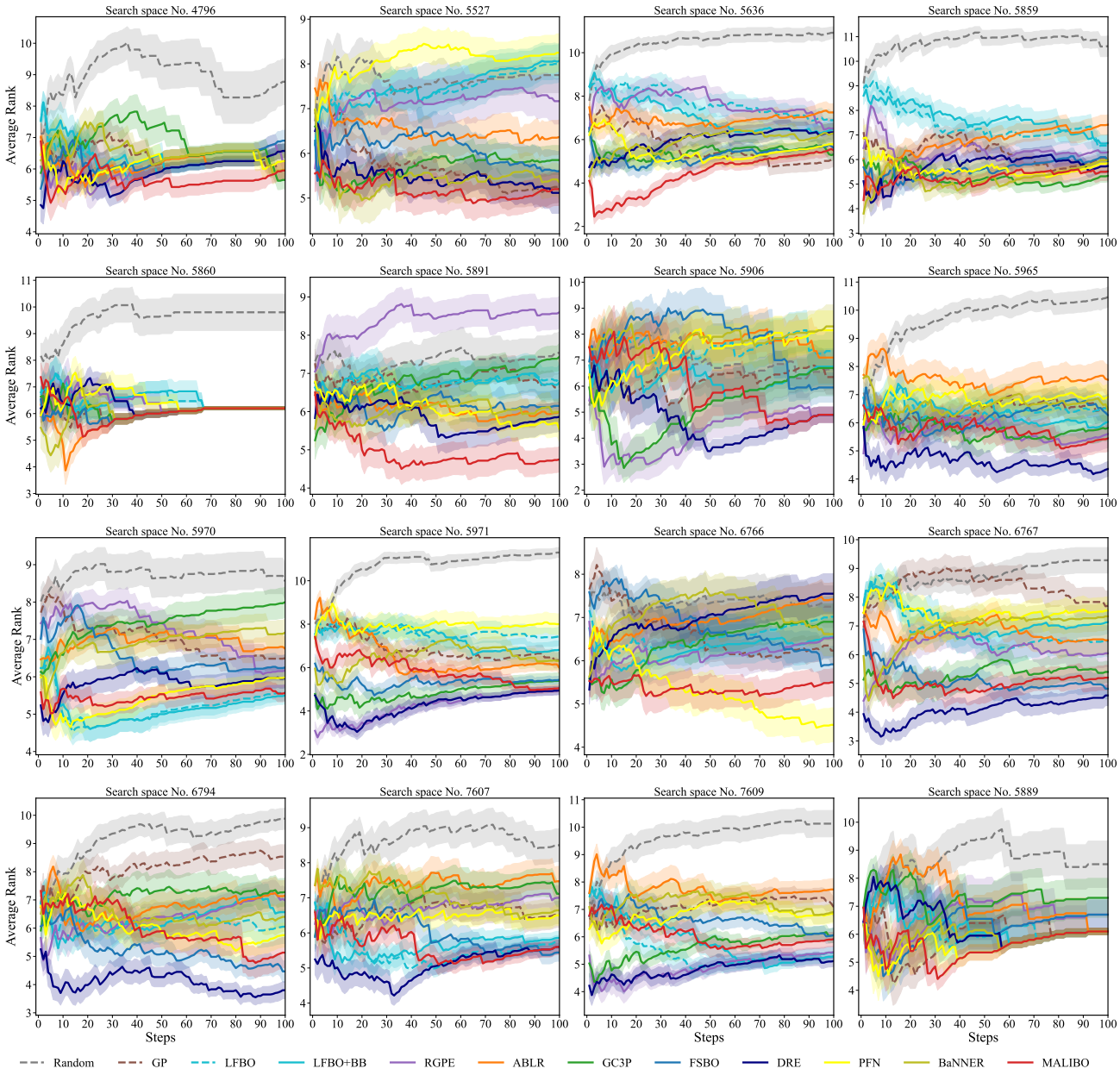


Figure 25. Average rank comparison of BO methods on HPO-B.

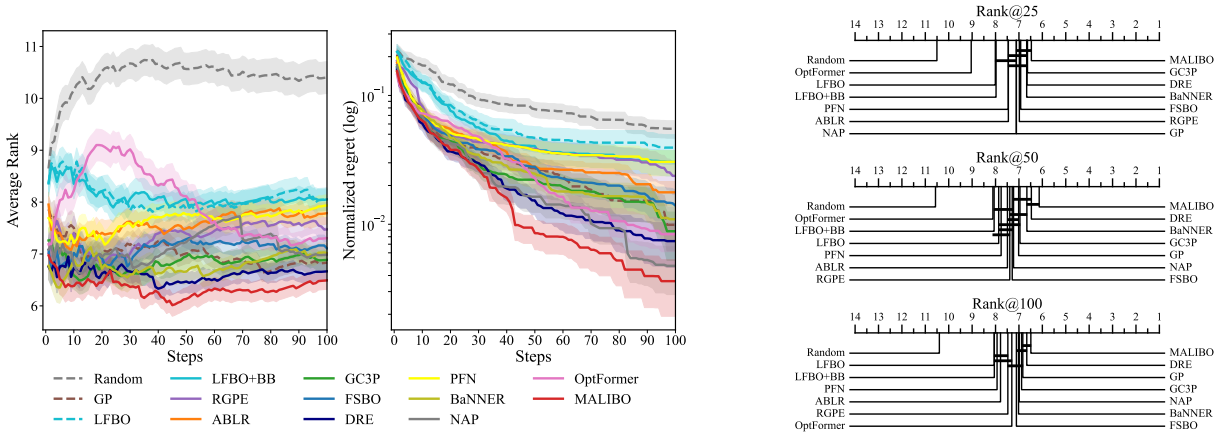


Figure 26. Aggregated comparisons of normalized regret and average ranks across 6 representative search spaces for BO methods on HPO-B.

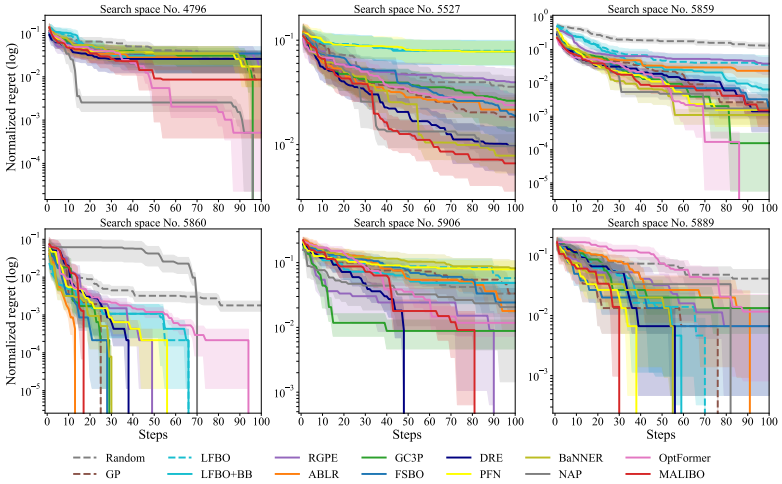


Figure 27. Normalized regret comparison of BO methods in 6 representative search space on HPO-B.

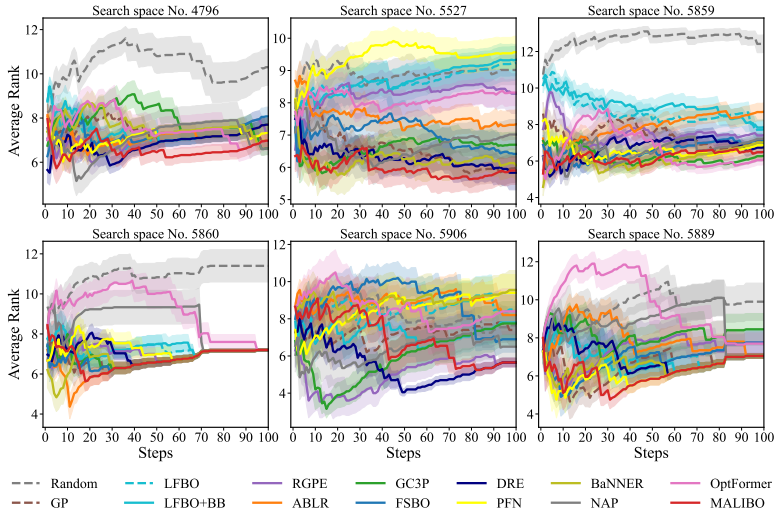


Figure 28. Average rank comparison of BO methods in 6 representative search space on HPO-B.

F. Step-through visualization

For illustration purposes, we provide step-through visualizations on a Forrester function. For details of the synthetic functions, we refer to Appendix H. We use the same meta-trained model for the visualizations as the one used in Appendix D.1 for the corresponding problem.

Sequential BO For the step-through visualization, the initial design is provided by the highest utility value of the mean predictions. After the first proposed query, we collect our the observations for the following 4 iterations using only the Thompson samples of the acquisition function. This is because we need to provide enough data to train and apply early stopping for our gradient boosting classifier. We provide the step-through visualization in Figure 29.

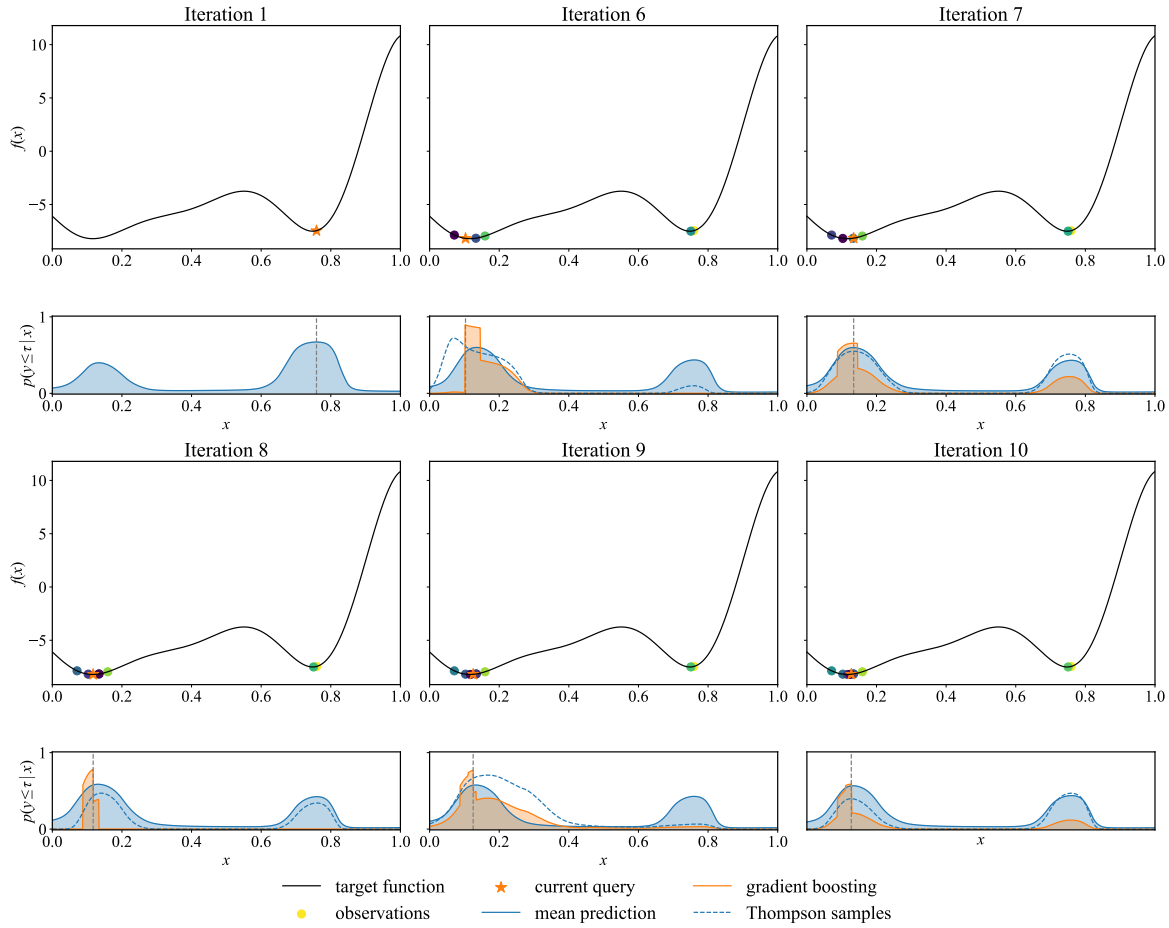


Figure 29. MALIBO optimizing a Forrester function. We show the mean prediction, the Thompson samples of the acquisition function and the gradient boosting prediction in the lower part of each sub-figure. At the first iteration, MALIBO picks the point with highest mean prediction of the acquisition function, which is often already close to the global optimum. Thereafter, we collect 4 more observations via the maximum prediction of a Thompson sample, in order to have sufficient data to train and apply early stopping for the gradient boosting model. Observations picked by Thompson samples show that MALIBO explores another location of interest on the left-hand side and also area close to the true optimum. With gradient boosting, the model is still able to explore the function and the predictions on non promising area are suppressed in later iterations.

Parallel BO with Thompson sampling After showing the step-through visualization for MALIBO, we try to showcase a preliminary experiments about extending MALIBO to parallel BO. We show a toy examples of synchronous parallel BO (Kandasamy et al., 2018) using MALIBO (TS) on the same function. To be specific, we use three Thompson samples as acquisition functions in each iteration, and evaluates the three proposed points for the next optimization step. We demonstrate that, MALIBO can be easily extended to parallel BO with the help of Thompson sampling.

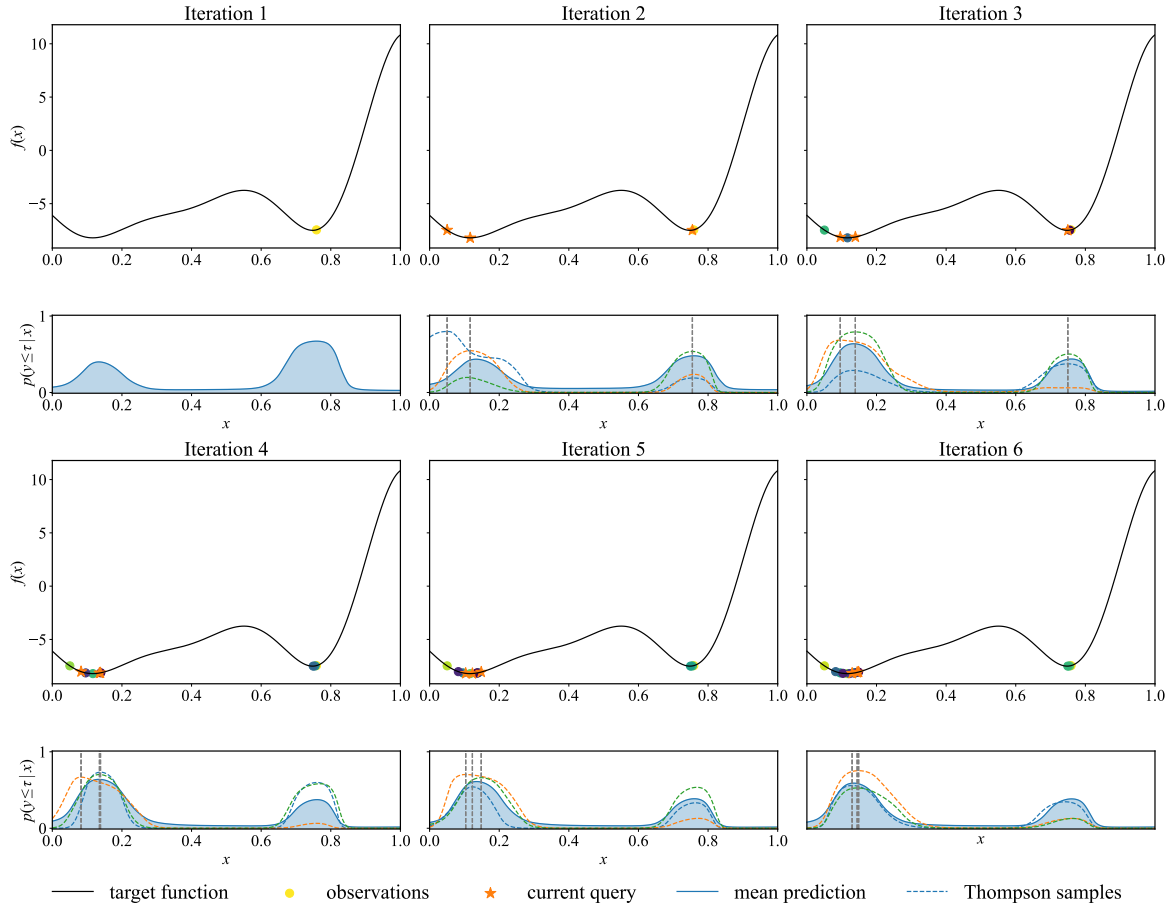


Figure 30. Synchronous parallel Thompson sampling using MALIBO (TS) to optimize a Forrester function. At every iteration, three samples are drawn as acquisition functions and utilized to the determine the next query points. In the first iteration, MALIBO (TS) already acquires three observations which cover both likely positions for the optimum. Subsequently, MALIBO (TS) exploits more often around the area where the true optimum is located. At the last iteration, all of the three Thompson samples have already been skew toward the left-hand side, which shows MALIBO (TS) converges to the correct region.

G. Experimental details

In this section, we explain the setups for all baselines we used in the experiments. We ran all baselines on 4 CPUs (Intel(R) Xeon(R) Gold 6150 CPU @ 2.70GHz) except for MetaBO, which requires more computation and we explain the details down below. Since most baselines are tested on similar AutoML tasks, we keep their hyperparameters as in their official implementations (if available) with minor modifications. The hyperparameters of MALIBO is selected by validating on the synthetic functions. We then fixed the selected hyperparameters for all experiments to ensure it is not overfit on a specific task. We will elaborate the experimental details in the following.

GP We use the SingleTaskGP implementation from BoTorch[3] with Matérn $5/2$ kernel and Expected Improvement (EI) as acquisition function.

LFBO Our implementation of LFBO is based on the official repository[4] from Song et al. (2022) and we use gradient boosting from scikit-learn (Pedregosa et al., 2011) as the classifier with the following settings: n_estimator $= 100$, learning rate $= 0.1$, min_samples_split $= 2$, min_samples_leaf $= 1$. For each problem, LFBO first randomly samples 10 observations to gather information and thereafter perform optimization using the classifier. For the threshold γ, which

[3] https://github.com/pytorch/botorch/
[4] https://github.com/lfbo-ml/lfbo

trade-off the exploration and exploitation, we set $\gamma = 1/3$ following Song et al. (2022) for all experiments. To maximize the resulting acquisition function, we use random search with $5,120$ samples following Tiao et al. (2021), where they show that, the acquisition function is usually non-smooth and discontinuous for decision trees based method and using random search is on par or even outperforms the more expensive alternative evolutionary algorithm.

LFBO+BB We extend LFBO to a meta-learning method with bounding box search space pruning (Perrone et al., 2019), which reduces the search space based on the promising configurations in the related tasks. Our implementation of the search space pruning technique is based on the open-source implementation in Syne Tune[5]. To construct the bounding box, we select the top-1 performing configurations from each related task, and truncate the search space according to the these configurations. We then apply LFBO to optimize the target task in the pruned search space.

RGPE Our implementation of Ranking-weighted GP Ensemble (RGPE) is based on Feurer et al. (2022). The key idea behind the algorithm is that, for optimization, the important information predicted by the surrogate model is not so much the function value $f(\mathbf{x})$ at a given input $\mathbf{x}$, but rather if the value $f(\mathbf{x})$ is larger or smaller relative to the function evaluated at other inputs. In other words, whether $f(\mathbf{x}) > f(\mathbf{x}')$ or vice versa. The algorithms propose to fit separate GPs on different tasks and use a ranking strategy to combine the model that fit the most on the target task with prior knowledge. Our implementation uses radidal basis function (RBF) kernel for each GP and Upper Confidence Bound (UCB) (Srinivas et al., 2010) with $\beta = 9.0$ as the acquisition function.

ABLR BO with multi-task adaptive Bayesian linear regression. Our implementation of ABLR is equivalent to a GP with 0 mean and a dot-product kernel with learned basis functions. We use a neural net (NN) with two hidden layers, each having 50 units, and Tanh activation as the basis functions. We train ABLR by optimizing the negative log likelihood (NLL) over NN weights and covariance matrix that define the dot-product kernel. In each iteration, ABLR is trained on all data including the meta-data as well as the observations from the target task using L-BFGS (Byrd et al., 1995).

GC3P We use the open-source implementation[6] for GC3P from Salinas et al. (2020). For each target function, GC3P first samples five candidates from a meta-learned NN model before building a task-specific Copula process.

BaNNER Similar to MALIBO, BaNNER uses a task-agnostic component to learn a feature transformation across tasks for the mean prediction and a task-specific component to predict the residual. The difference is that it uses Bayesian linear regression in the final layer, akin to the approach described in ABLR. We implemented the BaNNER-BLR-GP variant as detailed by Berkenkamp et al. (2021), which integrates an additive non-parametric GP model to account for residual errors. In our implementation, BaNNER utilizes a a three-layer ResFNN, each layer comprising 32 units, to learn the feature mapping function. We set the task embedding dimension to 16 and employed EI as the acquisition function.

FSBO We use the open-source implementation[7] for FSBO from Wistuba & Grabocka (2021). Its idea is similar to ABLR, which aims to learn a task-independent deep kernel surrogate and allow a task-dependent head for adaptation. The difference is that it frames meta-learning BO as a few-shot learning problem, which means during training, each batch from stochastic gradient ascent contains only data from one task. For the experiments, we use the default setting from the official implementation.

DRE We use the open-source implementation[8] for DRE from Khazi et al. (2023). As this implementation only provides pipelines and hyperparameter settings for the HPO-B benchmark, we confined our reporting to this benchmark to ensure a fair comparison.

PFN We use the open-source implementation[9] for PFN from Müller et al. (2023).

[5] https://github.com/awslabs/syne-tune/

[6] https://github.com/geoalgo/A-Quantile-based-Approach-for-Hyperparameter-Transfer-Learning

[7] https://github.com/releaunifreiburg/FSBO

[8] https://github.com/releaunifreiburg/DeepRankingEnsembles

[9] https://github.com/automl/PFNs4BO

NAP and OptFormer Due to the excessive training time, we use their results of HPO-B from the official repository[10].

MetaBO The training and evaluation of MetaBO was done based on the official implementation of Volpp et al. (2020). We followed the recommended hyperparameters and model architecture from the implementation with the following changes:

1. We extended the training horizon to 60 steps, which is longer than the ones used by Volpp et al. (2020). We chose the longer episode length to adapt to the higher dimensional space and the evaluation horizon we chose for the benchmarks. The value 60 was chosen as a compromise between the full evaluation length of 500 iterations and the resulting increase in training time due to the scaling of the GP used in the method.
2. We did not include the current time-step and total budget as features to the neural network policy due to poor performance on our benchmarks when including them.
3. Due to the small number of data sets, we estimated the GP hyperparameters with independent sub-samples of the meta-data sets, but otherwise following the procedure of Volpp et al. (2020). This effectively gives MetaBO access to more meta-data, but is consistent with the evaluation scheme described below.

Besides these changes to the method, we also employed a different evaluation scheme for MetaBO due to its high training cost. In contrast to the other meta-learning models that train in minutes on the meta-data using a single CPU, MetaBO required almost 2 hours, using one NVIDIA Titan X GPU and 10 Intel(R) Xeon(R) CPU E5-2697 v3 CPUs. This made the independent meta-training across the individual runs infeasible. To still include MetaBO into some of our benchmarks, we decided to train MetaBO once and reuse this model throughout the individual runs during the evaluation.

During the meta-training, MetaBO received the same number of samples per meta-task as the other methods, but the subsample was resampled for each training episode. While this gave MetaBO access to more meta-data compared to the other methods, we eliminated the risk of evaluating the method on a bad subsample of the data by chance. The advantage of effectively seeing more points of each meta-tasks should be considered when evaluating the early performance of MetaBO compared to the other methods. The evidently weak adaptation of MetaBO to new tasks dissimilar to the meta-data.

Based on the high meta-training cost of MetaBO and the relatively poor performance on NASBench201 and the HPOBench benchmarks, we decided to not include the method for the other evaluations, as scheme of leave-one-task-out validation would be too expensive and any other comparison would either benefit MetaBO or put it at a disadvantage rendering the results difficult to interpret.

MALIBO We use a Residual Feed Forward Network (ResFFN) (He et al., 2016) for learning the latent feature representation, with 4 hidden layers, each with 64 units. For the mean prediction layer $m(\cdot)$ and task-specific layer $h_{\mathbf{z}_t}(\cdot)$, we use a fully connected layer with 50 units for each. The resulting meta-leaning model has $22,359$ learnable parameters. We use ELU (Clevert et al., 2016) as the activation function in the network following Tiao et al. (2021). Similar to LFBO, we set the threshold $\gamma = 1/3$ and maximize the acquisition function using random search with $5,120$ samples..

During meta-training, we optimize the parameters in the network with the ADAM optimizer (Kingma & Ba, 2015), with learning rate $\text{lr} = 10^{-3}$ and batch size of $B = 256$. In addition, we apply exponential decay to the learning rate in each epoch with factor of 0.999. The model is trained for $2,048$ epochs with early stopping. For the regularization loss, we set the regularization factor $\lambda = 0.1$ in Equation (3) and follow the approach in Appendix C to estimate the coefficients λ_{KS} and λ_{Cov}. The resulting meta-training is fast and efficient and we show the training time as well as the amount of meta-data for each benchmark in Table 1.

In task adaptation, we optimize the task embedding for the target task using L-BFGS (Byrd et al., 1995), with learning rate $\text{lr} = 1$, maximal number of iterations per optimization step $\text{max\_iter} = 20$, termination tolerance on first order optimality $\text{tolerance\_grad} = 10^{-7}$, termination tolerance on function value/parameter changes $\text{tolerance\_change} = 10^{-9}$, $\text{history\_size} = 100$ and using strong-wolfe as line search method. After obtaining the model adapted on target task, we combine it with a gradient boosting classifier, which serves as a residual prediction model. We use the same setting for the gradient boosting as in LFBO, except that we use the meta-learned MALIBO classifier as the initial estimator. However, the gradient boosting classifier is trained only on the observations generated by the optimization process, which might lead to overfitting on limited amount of data during early iterations. Therefore, we apply early stopping to avoid such behavior.

[10] https://github.com/huawei-noah/HEBO/tree/master/NAP

Table 1. Meta-data and training time

Benchmark	# meta-data	Training time (approximate)
HPOBench	$1,536$	15 seconds
NASBench201	$1,024$	15 seconds
Branin	$32,768$	480 seconds
Hartmann3D	$131,072$	1,800 seconds

Specifically, we first estimate the number of trees that we need for training without overfitting. This is done by fitting a gradient boosting classifier with randomly chosen training data and validation data, which account for 70% and 30% of the whole data respectively. The resulting classifier estimates the number of trees that are needed to fit the partially observed data while offering good generalization ability. We then use it as our hyperparameter for the gradient boosting and train it on all observations.

H. Details of benchmarks

HPOBench

The hyperparameters for HPOBench and their ranges are demonstrated in Table 2. All hyperparameters are discrete and there are in total $66{,}208$ possible combinations. More details can be found in Klein & Hutter (2019).

Table 2. Configuration spaces for HPOBench

Hyperparameter	Range
Initial LR	$\{ 5 \times 10^{-4}, 1 \times 10^{-3}, 5 \times 10^{-3}, 1 \times 10^{-2}, 5 \times 10^{-2}, 1 \times 10^{-1} \}$
LR Schedule	{ cosine, fixed }
Batch size	$\{ 2^3, 2^4, 2^5, 2^6 \}$
Layer 1 Width	$\{ 2^4, 2^5, 2^6, 2^7, 2^8, 2^9 \}$
Activation	{ relu, tanh }
Dropout rate	$\{ 0.0, 0.3, 0.6 \}$
Layer 2 Width	$\{ 2^4, 2^5, 2^6, 2^7, 2^8, 2^9 \}$
Activation	{ relu, tanh }
Dropout rate	$\{ 0.0, 0.3, 0.6 \}$

NASBench201

The hyperparameters for NASBench201 and their ranges are summarized in Table 3. All hyperparameters are discrete and there are in total $15{,}625$ possible combinations. More details can be found in Dong & Yang (2020).

Table 3. Configuration spaces for NASBench201

Hyperparameter	Range
ARC 0	{ none, skip-connect, conv-1 $\times$ 1, conv-3 $\times$ 3, avg-pool-3 $\times$ 3 }
ARC 1	{ none, skip-connect, conv-1 $\times$ 1, conv-3 $\times$ 3, avg-pool-3 $\times$ 3 }
ARC 2	{ none, skip-connect, conv-1 $\times$ 1, conv-3 $\times$ 3, avg-pool-3 $\times$ 3 }
ARC 3	{ none, skip-connect, conv-1 $\times$ 1, conv-3 $\times$ 3, avg-pool-3 $\times$ 3 }
ARC 4	{ none, skip-connect, conv-1 $\times$ 1, conv-3 $\times$ 3, avg-pool-3 $\times$ 3 }
ARC 5	{ none, skip-connect, conv-1 $\times$ 1, conv-3 $\times$ 3, avg-pool-3 $\times$ 3 }

HPO-B

We use the HPO-B-v3 in our experiments and provide its description of search spaces in Table 4. More details can be found in Pineda-Arango et al. (2021).

Table 4. Description of the search spaces in HPO-B-v3. #HPs stands for the number of hyperparameters, #Evals. for the number of evaluations in a search space, while #DS for the number of datasets across which the evaluations are collected. The search spaces are named with the respective OpenML version number (in parenthesis).

Search Space	ID	#HPs	Meta-Train		Meta-Validation		Meta-Test	
			#Evals.	#DS	#Evals.	#DS	#Evals.	#DS
rpart.preproc (16)	4796	3	10694	36	1198	4	1200	4
svm (6)	5527	8	385115	51	196213	6	354316	6
rpart (29)	5636	6	503439	54	184204	7	339301	6
rpart (31)	5859	6	58809	56	17248	7	21060	6
glmnet (4)	5860	2	3100	27	598	3	857	3
svm (7)	5891	8	44091	51	13008	6	17293	6
xgboost (4)	5906	16	2289	24	584	3	513	2
ranger (9)	5965	10	414678	60	73006	7	83597	7
ranger (5)	5970	2	68300	55	18511	7	19023	6
xgboost (6)	5971	16	44401	52	11492	6	19637	6
glmnet (11)	6766	2	599056	51	210298	6	310114	6
xgboost (9)	6767	18	491497	52	211498	7	299709	6
ranger (13)	6794	10	591831	52	230100	6	406145	6
ranger (15)	7607	9	18686	58	4203	7	5028	7
ranger (16)	7609	9	41631	59	8215	7	9689	7
ranger (7)	5889	6	1433	20	410	2	598	2

The Quadratic Ensemble

The function for the quadratic ensemble is defined as:

$$f(x, a, b, c) = (a \cdot (x - b))^2 - c \qquad x \in [0, 1] \tag{30}$$

To form the ensemble, we choose the distribution for the parameters as:

$$a \sim \mathcal{U}(0.5, 1.5) \quad b \sim \mathcal{U}(-0.9, 0.9) \quad c \sim \mathcal{U}(-1, 1) \tag{31}$$

This distribution of parameters ensures that the search space contains the minimum of the quadratic function at $x^* = b$ with $f(x^*) = c$. The location of the optimum has a broad distribution over the function space, which is intended to highlight algorithms that learn the global structure of the ensemble rather than restricting on some small regions of interest.

The Forrester Ensemble

The original Forrester function (Sobester et al., 2008) is defined following:

$$f(x, a, b, c) = a \cdot (6x - 2)^2 \dot{\sin}(12x - 4) + b(x - 0.5) - c\,, \qquad x \in [0, 1] \tag{32}$$

The function has one local and one global minimum, and a zero-gradient inflection point in the domain $x \in [0, 1]$. To form the ensemble, we choose the distribution for the parameters as:

$$a \sim \mathcal{U}(0.2, 3) \quad b \sim \mathcal{U}(-5, 15) \quad c \sim \mathcal{U}(-5, 5) \tag{33}$$

Let $\tau = \{a, b, c\}$ and $p(\tau)$ is a three dimensional uniform distribution. The ranges are chosen around the usually used fixed values for the parameters, namely $a = 0.5$, $b = 10$, $c = -5$.

The Branin Ensemble

The function for the Branin ensemble is the following:

$$f(x, a, b, c) = a(x_2 - bx_1^2 + cx_1 - r) + s(1-t)\cos(x_1) + s\,, \qquad x_1 \in [-5, 10], x_2 \in [0, 15] \quad (34)$$

The distribution for the parameters are chosen as:

$$\begin{aligned} a &\sim \mathcal{U}(0.5, 1.5) \quad b \sim \mathcal{U}(0.1, 0.15) \quad c \sim \mathcal{U}(1.0, 2.0) \\ r &\sim \mathcal{U}(5.0, 7, 0) \quad s \sim \mathcal{U}(8.0, 12.0) \quad t \sim \mathcal{U}(0.03, 0.05) \end{aligned} \quad (35)$$

Let $\tau = \{a, b, c, r, s, t\}$ and $p(\tau)$ is a six dimensional uniform distribution. The ranges are chosen around the usually used fixed values for the parameters, namely $a = 1$, $b = 5.1/(4\pi^2)$, $c = 5/\pi$, $r = 6$, $s = 10$ and $t = 1/(8\pi)$.

The Hartmann3D Ensemble

The function for Hartmann3D (Dixon, 1978) ensemble reads:

$$\begin{aligned} &f(x, \alpha_1, \alpha_2, \alpha_3, \alpha_4) = \\ &\quad -\sum_{i=1}^{4} \alpha_i \exp\left(-\sum_{j=1}^{3} A_{i,j}(x_j - P_{i,j})^2\right) \qquad x \in [0, 1] \\ &\boldsymbol{A} = \begin{bmatrix} 3.0 & 10 & 30 \\ 0.1 & 10 & 35 \\ 3.0 & 10 & 30 \\ 0.1 & 10 & 35 \end{bmatrix} \quad \boldsymbol{P} = 10^{-4} \cdot \begin{bmatrix} 3689 & 1170 & 2673 \\ 4699 & 4387 & 7470 \\ 1091 & 8732 & 5547 \\ 381 & 5743 & 8828 \end{bmatrix} \end{aligned} \quad (36)$$

To form the ensemble, we choose the distribution for the parameters as:

$$\begin{aligned} \alpha_1 &\sim \mathcal{U}(0.0, 2.0) \quad \alpha_2 \sim \mathcal{U}(0.0, 2.0) \\ \alpha_3 &\sim \mathcal{U}(2.0, 4.0) \quad \alpha_4 \sim \mathcal{U}(2.0, 4.0) \end{aligned} \quad (37)$$

Let $\tau = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$ and $p(\tau)$ is a four dimensional uniform distribution.