Journal of Machine Learning Research 25 (2024) 1-83 Submitted 3/23; Revised 6/24; Published 7/24

Pre-trained Gaussian Processes for Bayesian Optimization

Zi Wang WANGZI@GOOGLE.COM
George E. Dahl GDAHL@GOOGLE.COM
Kevin Swersky KSWERSKY@GOOGLE.COM
Chansoo Lee CHANSOO@GOOGLE.COM
Zachary Nado ZNADO@GOOGLE.COM
Justin Gilmer GILMER@GOOGLE.COM
Jasper Snoek JSNOEK@GOOGLE.COM
Zoubin Ghahramani ZOUBIN@GOOGLE.COM
Google DeepMind

Editor: Marc Peter Deisenroth

Abstract

Bayesian optimization (BO) has become a popular strategy for global optimization of expensive real-world functions. Contrary to a common expectation that BO is suited to optimizing black-box functions, it actually requires domain knowledge about those functions to deploy BO successfully. Such domain knowledge often manifests in Gaussian process (GP) priors that specify initial beliefs on functions. However, even with expert knowledge, it is non-trivial to quantitatively define a prior. This is especially true for hyperparameter tuning problems on complex machine learning models, where landscapes of tuning objectives are often difficult to comprehend. We seek an alternative practice for setting these functional priors. In particular, we consider the scenario where we have data from similar functions that allow us to pre-train a tighter distribution a priori. We detail what pre-training entails for GPs using a KL divergence based loss function, and propose a new pre-training based BO framework named HyperBO. Theoretically, we show bounded posterior predictions and near-zero regrets for HyperBO without assuming the "ground truth" GP prior is known. To verify our approach in realistic setups, we collect a large multi-task hyperparameter tuning dataset by training tens of thousands of configurations of near-state-of-the-art deep learning models on popular image and text datasets, as well as a protein sequence dataset. Our results show that on average, HyperBO is able to locate good hyperparameters at least 3 times more efficiently than the best competing methods on both our new tuning dataset and existing multi-task BO benchmarks.

Keywords: Bayesian optimization, Gaussian processes, pre-trained models, transfer learning, hyperparameter tuning

©2024 Zi Wang, George E. Dahl, Kevin Swersky, Chansoo Lee, Zachary Nado, Justin Gilmer, Jasper Snoek and Zoubin Ghahramani.

License: CC-BY 4.0, see https://creativecommons.org/licenses/by/4.0/. Attribution requirements are provided at http://jmlr.org/papers/v25/23-0269.html.

Contents

1 Introduction **4**

2 Literature Review **7**

3 Background **8**

4 Problem Formulation **10**

5 Pre-training Gaussian Processes **11**
5.1 Pre-training Objective . . . 12
5.2 Empirical KL Divergence (EKL) . . . 13
5.2.1 Case Study: Observing Training Functions on the Same Inputs . . . 13
5.2.2 Approximation by Estimation . . . 13
5.2.3 Extensions to Generic Cases of Training Data . . . 15
5.2.4 Multiple "Matching-input" Datasets . . . 16
5.3 Negative Log Likelihood (NLL) . . . 16
5.3.1 NLL for Generic Cases of Training Data . . . 17
5.4 Computational Complexity . . . 17
5.5 Relations between EKL and NLL . . . 19
5.5.1 Conditions for the Equivalence of EKL and NLL . . . 19
5.5.2 Differences between EKL and NLL . . . 19

6 HyperBO: Bayesian Optimization with Pre-trained Gaussian Process Priors **20**
6.1 Posterior Inference and Acquisition Strategies . . . 21
6.2 Case Study: Functions with Finite Domains . . . 23
6.2.1 Bounding Pre-trained GP Posterior Predictions . . . 23
6.2.2 Theoretical Analyses on Regret Bounds . . . 24
6.3 Understanding HyperBO with Synthetic Examples . . . 25
6.3.1 Simple Function Domains . . . 25
6.3.2 Function Augmentation by Adding More Training Datapoints . . . 27

7 Experiments **28**
7.1 PD1: A New Hyperparameter Tuning Benchmark for Optimizing Deep Learning Models . . . 29
7.2 Description of All Compared Methods . . . 30
7.3 Reporting Aggregated Multi-task Results . . . 33
7.4 Results on the PD1 Offline Hyperparameter Tuning Tasks . . . 33
7.4.1 Holding Out Relevant Tasks . . . 34
7.4.2 Correlations between the Test NLL, Pre-training Loss and BO Performance 36
7.4.3 Effects of the Number of Training Datapoints . . . 36
7.4.4 Effects of the Number of Training Tasks . . . 38
7.5 Results on the PD1 Online Hyperparameter Tuning Tasks . . . 39
7.6 Results on the HPO-B Benchmark . . . 42
7.6.1 Pre-training on the Full HPO-B Dataset . . . 42

7.6.2 Pre-training on HPO-B with Reduced Training Data 44
7.6.3 The "Negative Transfer" Effect in HPO-B 45
7.7 Summary of Experiments 46

8 Discussion **46**
8.1 Fully Bayesian Interpretations of HyperBO 46
8.2 Important Extensions of HyperBO to Address Real-world Challenges of BO 48
8.3 Other Open Problems 49

9 Conclusion **50**

A Author Contributions **52**

B Derivations for EKL **52**

C Proofs for Theorem 5 and Theorem 6 **54**
C.1 Theorem 5: Bounding Pre-trained GP Posterior Predictions 54
C.2 Theorem 6: Bounding the Regrets of HyperBO 58
C.3 An Alternative Approach to Bound the Regret 63

D Experiment Details and More Results **65**
D.1 More Results on the PD1 Online Tuning Tasks 66
D.2 Impact of Acquisition Functions 66
D.3 Impact of Mean and Kernel Structures 66
D.4 More Results on HPO-B 70
D.5 MAF Implementation Details 71

1. Introduction

Bayesian optimization (BO) (Garnett, 2023) has been successfully applied in numerous real-world global optimization problems, ranging broadly from hyperparameter tuning (Snoek et al., 2012; Kotthoff et al., 2017) to chemical synthesis (Griffiths and Hernández-Lobato, 2020; Shields et al., 2021), drug discovery (Pyzer-Knapp, 2018), aerospace engineering (Lam et al., 2018), robotics (Drieß et al., 2017; Wang et al., 2017a) and more. However, in some scenarios, BO has been reported to under-perform naive strategies including random search (Li et al., 2017). While recent collective efforts have shown that "Bayesian optimization is superior to random search" (Turner et al., 2021), we seek more understanding on why BO works in some hands but not others.

Many successful BO applications benefit from expert knowledge on characteristics of the function to be optimized and hands-on experience with BO on similar tasks in the past. Such knowledge or experience can give intuitions about a functional form of the problem and thus specifications of a functional prior, e.g., a Gaussian process (GP) with squared exponential kernels for smoothness (Griffiths et al., 2008; Wilson et al., 2015). Sometimes people may be uncertain about their own understanding, and as a result they might choose to use a hierarchical model (Malkomes and Garnett, 2018; Cowen-Rivers et al., 2022; Snoek et al., 2012) or Bayesian neural nets (Springenberg et al., 2016), such that observed data can play a more important role in modeling. Despite having almost no information about a function, we can guess a generic prior from past experience with BO on other functions (Turner et al., 2021).

In the absence of any knowledge or experience, it is reasonable to consider all mathematical functions mapping inputs to outputs as equally likely. In this state of complete uncertainty, there is no single optimization algorithm that is guaranteed to outperform others (Schaffer, 1994). This limitation is commonly known as “no free lunch for optimization” (Wolpert and Macready, 1997). In the context of BO, it means no guarantee on performance if no informative prior is available. It is thus not surprising that we might encounter poor empirical performance (Schulz et al., 2016) when using BO without a well-specified prior. Theoretically, existing no-regret results only hold if model misspecification is well under control (Bogunovic and Krause, 2021; Berkenkamp et al., 2019).

Barriers of understanding on priors from a target domain and enough experience with BO can often turn away potential practitioners even within the machine learning (ML) community (Bouthillier and Varoquaux, 2020). For example, one of the most challenging domains for quantifying priors in BO is real-world hyperparameter tuning problems for modern deep learning models (e.g., ResNet50 from He et al., 2016) and large-scale datasets (e.g., ImageNet from Russakovsky et al., 2015). For those large models (He et al., 2016; Raffel et al., 2020; Brown et al., 2020), it is especially difficult to understand the landscapes of tuning objectives. Even for experts with relevant experience, it is hard to pin down what exactly this prior looks like, and researchers often have to study effects of hyperparameters one at a time (Sutskever et al., 2013; Zhang et al., 2019; Choi et al., 2019) given the complex structures of these problems. Since expert interventions on priors are almost unobtainable, use of BO has been hindered on these challenging but impactful tasks.

We would like to make BO methods more accessible by freeing practitioners from manually translating their own abstract beliefs into a quantitative Bayesian prior. We seek to automate the prior determination process by *pre-training GP priors* on data that are available on different but related tasks. Note that pre-training is also known as prior learning and can be considered a version of meta learning (Schmidhuber, 1995; Baxter, 1996; Minka and Picard, 1997). While the term is often used

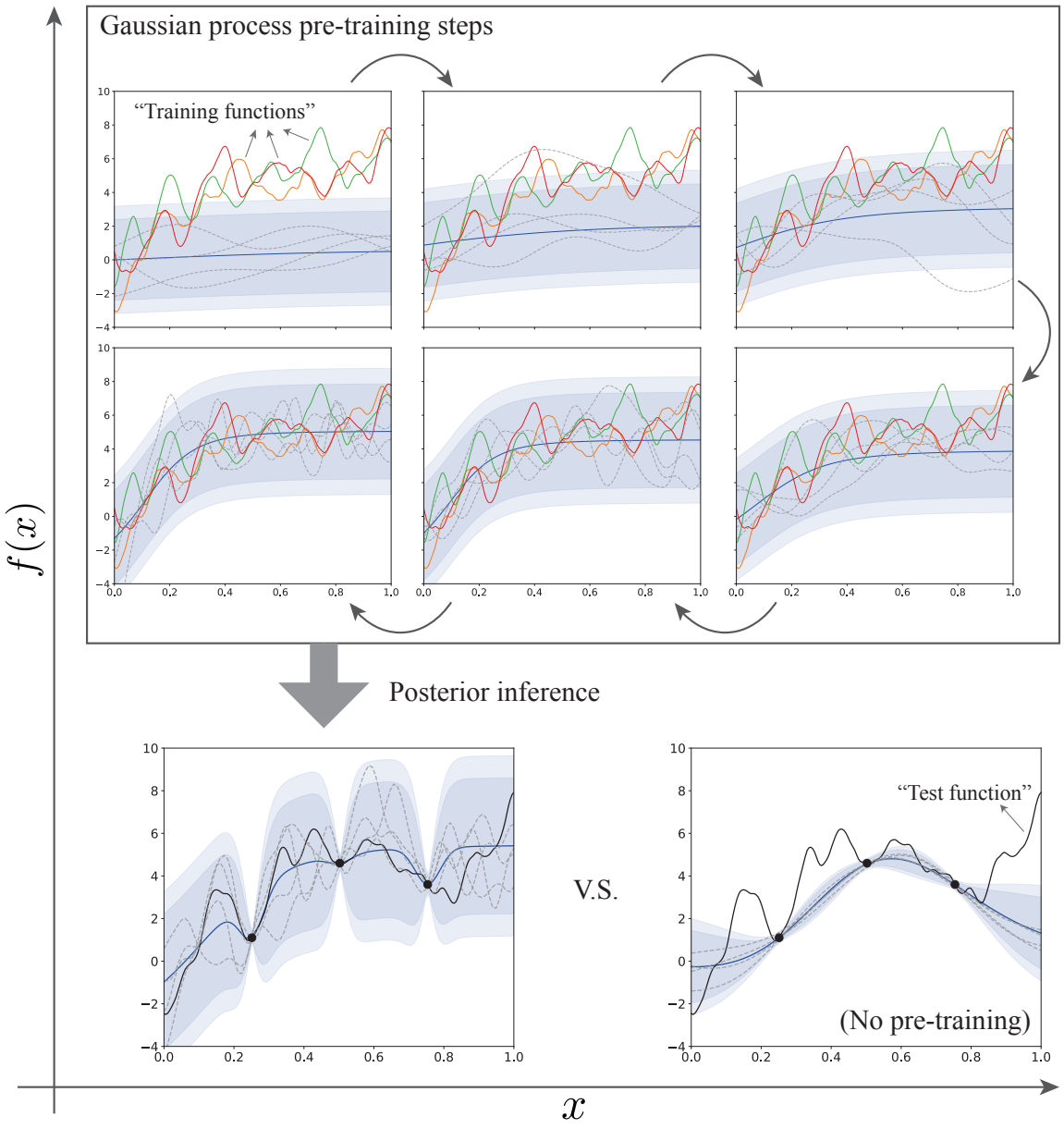

Figure 1: During pre-training, we optimize a Gaussian process (GP) such that it can gradually generate functions (illustrated as grey dotted lines) that are similar to the training functions. The similarity manifests in individual function values and correlations between function values indicated by smoothness and wiggliness. The blue line illustrates the mean function of the GP and the shaded areas are the 99% and 95% confidence intervals. For an unknown test function, we can derive a posterior conditioned on observed datapoints (illustrated as black dots) and the pre-trained GP prior. Compared to a GP fit to observations without pre-training, the pre-trained GP posterior captures the test function much better, which is a critical prerequisite for Bayesian optimization.

in deep learning, we contextualize *pre-training* for GPs in this work with a KL divergence based loss function, and use pre-trained GPs to bypass manual quantification of priors.

We hereby propose HyperBO: a BO framework with pre-trained GP priors. HyperBO is an enhancement of traditional BO methods without the requirement for practitioners to quantify their beliefs on functions. Instead, it is sufficient to specify which existing tasks are relevant and point to past evaluations on each of the functions corresponding to the existing tasks. These functions construct our training dataset for pre-training GPs. Figure 1 illustrates the iterative training steps to obtain a pre-trained GP prior and use it to derive posteriors for BO.

In essence, HyperBO empowers BO to overcome "no free lunch" and unlock the full potential of Bayesianism by creating informative priors. One of the key advantages of HyperBO is that we can guarantee success in terms of regret bounds under mild conditions, as long as the training functions can be viewed as samples from the unknown ground truth GP. Moreover, by replacing manual prior quantification with abstract identification of training functions, HyperBO streamlines the BO interface and allows easier use of BO for complex functions.

Empirically, we studied HyperBO on challenging modern ML tuning problems. To fill the vacancy of relevant "experience" for such problems, we collected PD1, a large multi-task hyperparameter tuning dataset, by training tens of thousands of configurations of near-state-of-the-art models (He et al., 2016; Raffel et al., 2020; Brown et al., 2020) on popular image and text datasets, as well as on a protein sequence dataset. With the PD1 tuning dataset for pre-training, we evaluated HyperBO on deep learning optimizer tuning problems, and on average, HyperBO achieved at least 3 times speedup than the best alternative method in terms of BO iterations needed to obtain best validation accuracy. Besides PD1, we also benchmarked the performance of HyperBO on HPO-B (Pineda-Arango et al., 2021), a collection of 16 multi-task BO benchmarks for classic machine learning models, and on average, HyperBO obtained at least 10 times speedup than competitive baselines.

HyperBO is related to our prior work (Wang et al., 2018b; Kim et al., 2019). Both Kim et al. (2019) and Wang et al. (2018b) were motivated by robot manipulation tasks where finite domains are sufficient. Wang et al. (2018b) considered compact domains but the only possible modeling choice is Bayesian linear regression due to the requirement of defining a GP with finite parameters. The key differences and unique contributions of this work are:

1. Significant new insights on a program based view of BO and how *pre-training* is consistent with the system of Bayesian belief reasoning (§3). Accordingly, we define a principled loss function and provide a unified view of two simple yet effective approximations for pre-training GPs as learned functional priors (§5).

2. Substantially relaxed assumptions on data availability and modeling choices. Our new framework is now compatible with any type of GP on both discrete and continuous input domains.

3. Entirely new application domains on challenging hyperparameter tuning tasks. Aside from tuning hyperparameters on classic machine learning models (Pineda-Arango et al., 2021), we applied our methods to tuning optimizer hyperparameters of modern deep learning models on popular image, text and protein sequence datasets.

4. Comprehensive analyses on the practicality of HyperBO. We provide new insights on how theoretical understandings carry to real-world experiments through case studies. Our empirical results show the notable advantage of HyperBO over strong baseline methods.

5. We open-sourced the first large multi-task hyperparameter tuning dataset for modern deep learning models. We spent roughly 12,000 machine-days to collect hyperparameters evaluations by training tens of thousands of configurations of near-state-of-the-art models on various scales of data, ranging from millions of images to billions of words. Together with our open-sourced code for HyperBO, the released dataset ensures the reproducibility of our work[1]. More importantly, the dataset provides a realistic benchmark for multi-task BO, with open opportunities to explore detailed metrics for each training step and other auxiliary information.

Next, we discuss related work in §2, explain foundational concepts of BO in §3, formulate our problem in §4 and introduce the core GP pre-training method in §5. In §6, we present our HyperBO framework for the black-box function optimization. To understand the implications of substituting the ground truth with a pre-trained GP model, we provide theoretical insights on the asymptotic properties of the pre-trained model for posterior inference, and show regret bounds to explain when it is a good idea to use the pre-trained model for BO. In §7, we provide empirical evidence showing promising results of HyperBO for real-world black-box function optimization tasks. Finally, we discuss fully Bayesian interpretations and open problems of HyperBO in §8, and conclude in §9.

2. Literature Review

There is a rich literature of innovative methodologies to improve the efficiency of BO given related tasks or additional context. Here we discuss the most closely related work and explain why these do not solve the specific scenario which we envision. Specifically, our goal is a methodology that is scalable enough to share information across thousands of tasks, each with potentially hundreds of observations, such as in the context of a large BO service or library.

Pre-training and prior learning is directly related to meta learning, learning to learn and learning multiple tasks (Schmidhuber, 1995; Baxter, 1996; Minka and Picard, 1997; Caruana, 1997). Such meta learning and multi-task learning ideas can be naturally used to learn a GP prior (Rasmussen and Williams, 2006, Chapter 5). We use the word pre-training to refer to supervised pre-training, which is a general approach in the deep learning community (Girshick et al., 2014; Donahue et al., 2014; Devlin et al., 2019) to transfer knowledge from prior tasks to a new task. The same as pre-training deep features on a variety of tasks, Wang et al. (2018b) proposed to learn a GP prior by learning the basis functions, treating the independent function outputs as individual heads of a neural network. More recently, there has been theoretical advancement in the PAC-Bayesian framework to understand meta learning (Rothfuss et al., 2021) for GPs.

Several methods, including that which HyperBO extends, refer to their method as "meta BO" (Wang et al., 2018b; Volpp et al., 2020). In this work we use the term *meta BO* more generally to refer to the class of BO methods that use data from existing tasks to optimize a new task. Since standard BO is a learning process, it is consistent to call those methods meta BO methods given that they learn how to learn. Under this viewpoint, meta BO approaches also include multi-task BO (Swersky et al., 2013; Yogatama and Mann, 2014; Poloczek et al., 2017) and transfer learning BO methods, e.g., based on contextual GPs (Krause and Ong, 2011; Bardenet et al., 2013; Poloczek et al., 2016), quantiles (Salinas et al., 2020) or ensembles of GPs (Feurer et al., 2018; Wistuba et al., 2018). Some meta BO methods have also been studied for hyperparamter tuning tasks in machine learning (Feurer et al., 2015; Salinas et al., 2020).

1. Both open-sourced code and dataset are available at `https://github.com/google-research/hyperbo`.

To enable meta learned models to transfer knowledge from prior tasks to a new task in BO, assumptions need to be made to capture the connections among tasks. For both multi-task and contextual BO methods, such as Krause and Ong (2011); Swersky et al. (2013); Tighineanu et al. (2022), the connections are modeled directly through computing the similarities between tasks. These approaches typically scale cubically in both the number of tasks and observations in each task, meaning that they cannot gracefully scale across both without heavy approximations. When assuming that all inputs are equal across tasks, multi-task BO (Swersky et al., 2013) can be sped up using a Kronecker decomposition of the kernel to a task kernel and an input kernel which can be inverted separately; a similar assumption is made by Wang et al. (2018b). In comparison, HyperBO establishes the connections among tasks by positing a shared prior which renders the tasks conditionally independent. As a result, HyperBO scales linearly in the number of tasks (see §5.4), which facilitates efficient model pre-training.

Motivated by robot learning problems, Kim et al. (2017, 2019) started a different thread of meta BO literature, with a goal to transfer knowledge among robot manipulation tasks. Each task is a BO problem that optimizes a scoring function by sequentially selecting search strategies from a finite set. Kim et al. (2017, 2019) noted that the similarities among tasks are very difficult to model, since a slight change in the state can completely change the function landscape. To address this issue, they introduced a simple but elegant approach: estimating the correlations between scores of different search strategies; i.e., modeling the similarities between inputs as opposed to tasks.

Wang et al. (2018b) provided regret bounds for Kim et al. (2017, 2019) and extended it to Bayesian linear regression with neural net basis functions. Similar ideas were developed by Perrone et al. (2018); Wistuba and Grabocka (2021) for tuning the hyperparameters of machine learning models. Wang et al. (2018b) can be viewed as a generalization of these approaches in that Perrone et al. (2018) and Wistuba and Grabocka (2021) only use zero means. As shown in Kim et al. (2017, 2019), a flexible mean function is important for learning the initial datapoints to acquire. Wistuba and Grabocka (2021) overcomes this initialization issue by using a data-driven evolutionary algorithm to warm-start the initialization. We opt to parameterize a flexible mean function using a neural network, allowing for end-to-end optimization.

Although different terms are used, Wang et al. (2018b) and Perrone et al. (2018) concurrently proposed the idea of learning parameters of GP priors from multi-task datasets, while Wang et al. (2018b) was the first to clarify the assumptions of conditionally independent multi-task functions and show regret bounds for BO with an unknown GP prior.

Our proposed pre-training objectives are related to the objective functions in variational inference for approximating GPs (Titsias, 2009; Burt et al., 2020). Our key idea on pre-training is to minimize the KL divergence between the unknown ground truth GP and an approximate. On the other hand, variational inference in functional spaces (Sun et al., 2019; Burt et al., 2020) generally considers the KL divergence between an approximate and a posterior, which aims to improve computational efficiency for posterior predictions given large scale observations. While there are significant differences in goals and methods, the objective functions all boil down to the KL divergence for functional distributions. More details on our method can be found in §5.

3. Background

While there are other interpretations, we take the viewpoint of *artificial intelligence* and consider *Bayesian optimization* (BO) as a study of how an intelligent machine optimizes a numerical function:

making sequential decisions on data acquisition by reasoning about the machine's *posterior beliefs* on the function. The beliefs are expressed as *Bayesian* probabilities, which root in logic, common sense and rational reasoning about plausibility (Jaynes, 2003). Decisions on data acquisition involve choosing the inputs to query the function and observing their corresponding outputs.

As a mathematical tool, a popular version of BO is *Gaussian process (GP) optimization* (Srinivas et al., 2010; Contal et al., 2014; Wang et al., 2016a), which uses *GPs* as the Bayesian beliefs on the function. Popular decision making criteria on data acquisition include probability of improvement (Kushner, 1964), expected improvement (Močkus, 1974), upper confidence bound (Auer, 2002), entropy search (Hennig and Schuler, 2012), etc.

BO has been used to optimize expensive black-box functions and to solve experimental design problems. To understand how to apply BO to real-world applications, we introduce *the practitioner*, e.g., a scientist or an engineer, who delegates decision making to the intelligent machine. To take over the optimization process, the machine requires the practitioner to assign a prior on information related to the function.

The prior used by the machine should reflect the practitioner's understanding, i.e., the practitioner's posterior belief, about the function based on their past experience with related functions. Assigning such priors typically requires the practitioner to quantitatively encode their own belief in machine languages, which is not always clear given the stark difference between how humans think and how machines operate.

Writing a program to assign the prior. The practitioner's belief is based on their past experience, or more specifically, the data they observed on functions relevant to, but not necessarily the same as, the black-box function of interest.

In this work, we seek to circumvent the hurdle of manually specifying the prior by writing a prior assignment program that works with the past observed data directly. Thus, the practitioner only needs to identify the data (partitioned to observations on different functions) that is related to the function they would like to optimize. This program takes over the task of prior assignment, which has a contract to take in user specified data as the input and output a probability distribution consistent with the "ground truth" practitioner's belief on the function. Note that the practitioner is typically non-Bayesian and their belief on the function does not necessarily reflect a posterior.

A wrapper over the Bayesian component. We have introduced two entirely separate programs: one does BO given a prior, and the other produces a probability distribution to match the underlying prior on the function. The former is a reasoning and decision making process governed by Bayes rules. The latter serves as the prior assignment program which, in this work, is not fully Bayesian. A system composed of these two programs bypasses the need to define a specific prior in BO and instead, derives the prior from data before doing any Bayesian inference or reasoning.

Pre-training as the prior assignment program. The prior assignment program needs to train a probabilistic model on data observed over different functions and assign the trained model as the prior for the black-box function of interest. We use *pre-training* to describe this program, as it is fundamentally the same practice as (supervised) pre-training in the deep learning literature: a model is trained on a range of tasks and the learned feature representations (i.e., basis functions) are preserved for unseen tasks for fine-tuning.

Note that "pre-training" has other names in the literature. Baxter (1996) explained learning a prior as "bias learning" from a Bayesian viewpoint. Minka and Picard (1997) more explicitly described it as learning bases of a probabilistic model from a dataset of tasks (each with some datapoints) and

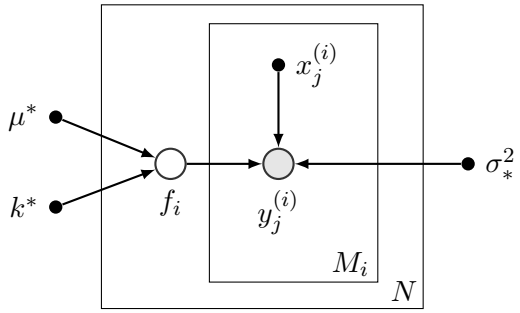

Figure 2: The generating process of our training data: for each $i \in [N]$, $f_i \sim \mathcal{GP}(\mu^*, k^*)$, and for each $j \in [M_i]$, $y_j^{(i)} \sim \mathcal{N}\left(f_i(x_j^{(i)}), \sigma_*^2\right)$, where mean function μ^*, kernel function k^* and noise variance σ_*^2 are unknown.

then applying the learned model as the prior for an unseen task. For ease of understanding, we use the term "pre-training" in this paper.

4. Problem Formulation

We follow the *Gaussian process (GP) optimization* paradigm: given a real-valued function f defined over a compact, hyper-rectangular space $\mathfrak{X} \subset \mathbb{R}^d$, we seek an $x \in \mathfrak{X}$ that maximizes f with as few evaluations on f as possible.

Our assumptions are similar to Minka and Picard (1997), where our training dataset is a set of *i.i.d.* sets of non *i.i.d.* datapoints, i.e., a dataset consisting of sets of observations on **training functions** $f_1, \cdots, f_N$. Assumption 1 emphasizes that our training functions and test functions are all *i.i.d.* samples from an unknown GP. Assumption 2 describes that observations are perturbed by *i.i.d.* Gaussian noises with unknown variance.

Assumption 1. *There exists a non-degenerate GP $\mathcal{GP}(\mu^*, k^*)$ with **unknown** mean function $\mu^* : \mathfrak{X} \to \mathbb{R}$ and **unknown** kernel $k^* : \mathfrak{X} \times \mathfrak{X} \to \mathbb{R}$, such that the training functions $f_1, \cdots, f_N$ and the **test function** f are all i.i.d. samples from $\mathcal{GP}(\mu^*, k^*)$.*

Assumption 2. *There exists a Gaussian distribution $\mathcal{N}(0, \sigma_*^2)$ with **unknown** variance $\sigma_*^2 \in \mathbb{R}^+$, such that for any function $g \sim \mathcal{GP}(\mu^*, k^*)$ and any input $x \in \mathfrak{X}$, the observed function value y is perturbed by i.i.d. additive Gaussian noise $\mathcal{N}(0, \sigma_*^2)$, i.e., $y \sim \mathcal{N}(g(x), \sigma_*^2)$.*

We use $[n]$ to denote $\{1, \cdots, n\}, \forall n \in \mathbb{Z}^+$. Let M_i be the number of observations we have for function f_i where $i \in [N]$. We use $y \sim \mathcal{N}(g(x), \sigma_*^2)$ as a short hand to describe the conditional distribution for $y \mid g$. Provided input $x_j^{(i)} \in \mathfrak{X}, i \in [N], j \in [M_i]$, the observed function value is $y_j^{(i)} \sim \mathcal{N}\left(f_i(x_j^{(i)}), \sigma_*^2\right)$ by Assumption 2.

Taken together, the collection of ***sub-datasets*** $D_{f_i} = \{(x_j^{(i)}, y_j^{(i)})\}_{j=1}^{M_i}$ constructs the ***training dataset*** $D_N = \{D_{f_i}\}_{i=1}^N$. Figure 2 shows the graphical model that illustrates the generating process of the training dataset D_N. Note that we explicitly assume that the ***ground truth*** μ^*, k^*, σ_*^2 exist but they are all ***unknown***.

Metrics. For simplicity, we focus on sequential evaluations on the *test function* f where only one input is chosen for evaluation in each iteration.

For T iterations of Bayesian optimization (BO) on function f, we accumulate a set of observations $D_f = \{(x_t, y_t)\}_{t=1}^T$, where $y_t \sim \mathcal{N}\left(f(x_t), \sigma_*^2\right)$ and T is a positive integer.

In hindsight, we can evaluate the quality of BO using the *simple regret* metric: $R_T = \max_{x \in \mathfrak{X}} f(x) - f(\hat{x})$, where $\hat{x}$ is the recommended best input.

There are various ways of setting $\hat{x}$ based on the observations D_f or the posterior of function f. In this work, we use the input that achieves the best evaluation: $\hat{x} = x_{T'}; T' = \arg\max_{\tau \in [T]} y_\tau$.

Notations of models. We use $\mathcal{GP}(\mu, k \circ \sigma^2)$ to denote a GP model with mean function μ and a kernel function with perturbed diagonal terms:

$$k \circ \sigma^2(x_j, x_{j'}) = k(x_j, x_{j'}) + \mathbb{1}_{j \equiv j'} \sigma^2,$$

where j and j' are indices of the inputs.

We also use the short-hands $k(x) = k(x, x)$ and $k \circ \sigma^2(x) = k(x, x) + \sigma^2$ for simplicity.

For any collection of inputs $\boldsymbol{x} = [x_j]_{j=1}^m \in \mathbb{R}^{m \times d}$ and an input $x' \in \mathbb{R}^d$, we denote the column vector of mean function values as $\mu(\boldsymbol{x}) = [\mu(x_j)]_{j=1}^m \in \mathbb{R}^m$, the column vector of kernel function values between $\boldsymbol{x}$ and x' as $k(\boldsymbol{x}, x') = [k(x_j, x')]_{j=1}^m \in \mathbb{R}^m$, and the Gram matrix as $k \circ \sigma^2(\boldsymbol{x}) = [k(x_j, x_{j'})]_{j \in [m], j' \in [m]} + I\sigma^2 \in \mathbb{R}^{m \times m}$.

Remarks. In the language of §3, the practitioner's belief, a.k.a. the unknown ground truth prior of the intelligent machine, is $\mathcal{GP}(\mu^*, k^* \circ \sigma_*^2)$, which describes the probability distributions of functions and observation noise. The contract of the prior assignment program is to take in as input *training dataset* D_N and output a *pre-trained GP* $\mathcal{GP}(\hat{\mu}, \hat{k} \circ \hat{\sigma}^2)$. Ideally, the pre-trained model should be very close to the ground truth prior. The intelligent machine can then use the pre-trained $\mathcal{GP}(\hat{\mu}, \hat{k} \circ \hat{\sigma}^2)$ as the "prior distribution" in a BO program for test function f.

Example 4.1. *In machine learning hyperparameter tuning applications, the task is to find the best configuration of hyperparameters to train a specific machine learning model on a particular dataset, e.g., training a ResNet (He et al., 2016) on ImageNet (Russakovsky et al., 2015). For each combination of a machine learning model and a dataset, there is an underlying function mapping from configurations of hyperparameters to an evaluation metric such as the error rate. The training functions $f_1, \cdots, f_N$ correspond to those functions that the practitioner observed in the past. The test function f maps from configurations of hyperparameters to the evaluation metric of a new combination of a model and a dataset.*

5. Pre-training Gaussian Processes

In this section, we will describe our KL divergence based objective for pre-training Gaussian processes (GPs). We will introduce two approximations for the objective. First is an estimator that we call the empirical KL (EKL) loss. The second reduces to the negative log-likelihood (NLL) loss. These have different strengths depending on the property of the data. When each task has an observation at each input, EKL can naturally learn and make use of the correlations between function values. We find that this improves its empirical performance. NLL is more flexible, and useful for cases where the observations are made at different locations across tasks.

5.1 Pre-training Objective

We use the KL divergence between the ground truth $\mathcal{GP}(\mu^*, k^* \circ \sigma_*^2)$ and a model $\mathcal{GP}(\mu, k \circ \sigma^2)$ as the loss function. By Theorem 1 of Sun et al. (2019), our loss function is

$$\mathcal{L}(\mu,\, k \circ \sigma^2) \coloneqq D_{\text{KL}}\Big(\mathcal{GP}(\mu^*, k^* \circ \sigma_*^2), \mathcal{GP}(\mu, k \circ \sigma^2)\Big) \tag{1}$$

$$= \sup_{\boldsymbol{x} \subseteq \mathfrak{X}, \text{card}(\boldsymbol{x}) < \infty} D_{\text{KL}}\Big(\mathcal{N}(\mu^*(\boldsymbol{x}), k^* \circ \sigma_*^2(\boldsymbol{x})), \mathcal{N}(\mu(\boldsymbol{x}), k \circ \sigma^2(\boldsymbol{x}))\Big). \tag{2}$$

In Eq. 2, $\boldsymbol{x}$ is a collection of inputs from $\mathfrak{X} \subset \mathbb{R}^d$ and $\text{card}(\boldsymbol{x})$ is the cardinality of $\boldsymbol{x}$. While it is often intractable to compute this loss function (Burt et al., 2020; Sun et al., 2019), it is natural to approximate the loss by truncating $\mathfrak{X}$ to a finite set of inputs (Sun et al., 2019). If the domain $\mathfrak{X}$ is finite, the loss in Eq. 1 becomes the KL divergence between two multivariate Gaussian distributions, and the supremum in Eq. 2 is obtained by setting $\boldsymbol{x} \equiv \mathfrak{X}$.

Our goal is to obtain the "pre-trained" GP model, $\mathcal{GP}(\hat{\mu}, \hat{k} \circ \hat{\sigma}^2)$, by minimizing the loss function subject to positive definite kernel and positive noise variance; that is,

$$\hat{\mu},\, \hat{k} \circ \hat{\sigma}^2 \;=\; \underset{\mu,\, k \circ \sigma^2}{\arg\min} \quad \mathcal{L}(\mu,\, k \circ \sigma^2) \tag{3}$$

$$s.t. \quad k > 0, \sigma^2 > 0$$

Without loss of generality, we assume there is one $\arg\min$ solution[2] to the minimization problem in Eq. 3. We slightly abuse notation by minimizing the loss over the mean function μ and perturbed kernel $k \circ \sigma^2$ without specifying their search spaces. In practice, the search spaces for these functions depend on specifications made by the practitioner. For example, we may try to minimize the loss by comparing its values on two different mean functions, e.g., $\mu(x) = |x|$ and $\mu(x) = |x|^2$, where the search space for μ contains two functions.

More generally, minimizing the loss in Eq. 1 can involve searching over function structures and/or optimizing the parameters of functions (Malkomes et al., 2016; Malkomes and Garnett, 2018). For high dimensional problems, we might prefer additive kernels for interpretability, and we can learn the additive structure as well (Wang et al., 2017b; Gardner et al., 2017; Rolland et al., 2018). For structured inputs, one may adopt specialized kernels, e.g., graph kernels (Vishwanathan et al., 2010), convolutional kernels (Van der Wilk et al., 2017), etc. Note that it is not necessary to require the mean function μ or perturbed kernel $k \circ \sigma^2$ to be parametric. For example, they can be specified with memory based machine learning models (Russell and Norvig, 2003; Daelemans et al., 2005).

If we use parametric mean function μ and perturbed kernel $k \circ \sigma^2$ with fixed structures, we only need to optimize their parameters. Wistuba and Grabocka (2021) proposed to use the Adam optimizer (Kingma and Ba, 2015); Wang et al. (2018b) suggested solving linear systems; and Perrone et al. (2018) recommended L-BFGS (Liu and Nocedal, 1989). The choice of optimizers may also depend on the parametric form. For our experiments, we defined flexible search spaces of functions using neural networks and constructed positive definite kernels on encoded representations of inputs. More details can be found in §7.

Now that it is clear there exist methods to optimize over spaces of functions, we investigate a more pressing issue on our objective in Eq. 3: we do not know the ground truth model $\mathcal{GP}(\mu^*, k^* \circ \sigma_*^2)$ and cannot compute the loss function.

2. The solution depends on the space of functions to optimize the loss function over, and it is possible that a minimum does not exist. However, for practical engineering setups and choices of models, it is reasonable to assume finding one (approximate) solution is possible.

In §5.2, we introduce EKL that directly computes the KL divergence by estimating a multivariate Gaussian distribution induced by $\mathcal{GP}(\mu^*, k^* \circ \sigma_*^2)$. In §5.3, we use NLL: an expansion of the KL divergence to simulate the loss with training functions. We analyze the computational complexity of the approximated loss functions in §5.4. In §5.5, we explain the relations between EKL and NLL.

5.2 Empirical KL Divergence (EKL)

The key idea of the EKL approximation is to compute the KL divergence between an empirical estimate of the ground truth and our model. Thus, it is possible to directly compute the KL divergence by manipulating the training data.

5.2.1 CASE STUDY: OBSERVING TRAINING FUNCTIONS ON THE SAME INPUTS

For simplicity, we first consider the case where the training dataset $D_N = \{D_{f_i}\}$ is a "matching-input" dataset, $D_{f_i} = \{(x_j, y_j^{(i)})\}_{j=1}^M$, where M is the number of shared inputs across N training functions.

Dataset D_N is composed of queries over the training functions $f_1, \cdots, f_N$ at the same set of input locations $\boldsymbol{x} = [x_j]_{j=1}^M \in \mathbb{R}^{M \times d}$. We can re-organize the datapoints in D_N as follows.

	f_1	$\cdots$	f_i	$\cdots$	f_N
x_1	$y_1^{(1)}$	$\cdots$	$y_1^{(i)}$	$\cdots$	$y_1^{(N)}$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$
x_j	$y_j^{(1)}$	$\cdots$	$y_j^{(i)}$	$\cdots$	$y_j^{(N)}$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$
x_M	$y_M^{(1)}$	$\cdots$	$y_M^{(i)}$	$\cdots$	$y_M^{(N)}$

$$\boldsymbol{y}_i = \begin{bmatrix} y_1^{(i)} \\ \vdots \\ y_M^{(i)} \end{bmatrix}$$

For each training function f_i, we have M observations which correspond to entries of a column in the illustration, i.e.,$\boldsymbol{y}_i = [y_j^{(i)}]_{j=1}^M \in \mathbb{R}^M$. Given that each function $f_i \overset{i.i.d.}{\sim} \mathcal{GP}(\mu^*, k^*)$ and the observations $y_j^{(i)} \overset{i.i.d.}{\sim} \mathcal{N}(f_i(x_j), \sigma_*^2)$, we have $\boldsymbol{y}_i \overset{i.i.d.}{\sim} \mathcal{N}(\boldsymbol{\mu}^*, \Sigma^*)$, where the mean vector is $\boldsymbol{\mu}^* = \mu^*(\boldsymbol{x})$ and the covariance matrix is $\Sigma^* = k^* \circ \sigma_*^2(\boldsymbol{x})$. The distribution $\mathcal{N}(\boldsymbol{\mu}^*, \Sigma^*)$ captures the marginals of the ground truth $\mathcal{GP}(\mu^*, k^* \circ \sigma_*^2)$.

5.2.2 APPROXIMATION BY ESTIMATION

We perform a two-step approximation of the loss function in Eq. 1: (1) estimate the marginal ground truth distribution and (2) approximate the KL divergence.

From the observations on all training functions, we can estimate the unknown mean vector $\boldsymbol{\mu}^*$ and covariance matrix Σ^*. By concatenating the columns $\boldsymbol{y}_i$ together horizontally, we obtain a matrix of observations on all training functions: $Y = [\boldsymbol{y}_i]_{i=1}^N \in \mathbb{R}^{M \times N}$.

In this work, we adopt maximum likelihood estimation (MLE) and get the estimated mean vector and covariance matrix as follows,

$$\tilde{\boldsymbol{\mu}} = \frac{1}{N} Y 1_N \in \mathbb{R}^M \quad \text{and} \quad \tilde{\Sigma} = \frac{1}{N}(Y - \tilde{\boldsymbol{\mu}} 1_N^\top)(Y - \tilde{\boldsymbol{\mu}} 1_N^\top)^\top \in \mathbb{R}^{M \times M}, \tag{4}$$

where 1_N is a column vector of size N filled with 1s. Other kinds of estimators can be used to replace MLE, such as the unbiased sample mean and covariance estimator.

The following *empirical KL divergence* (EKL) approximates the loss function (Eq. 1) with an empirical estimation of the ground truth model,

$$\begin{aligned}\mathcal{L}(\mu,\, k \circ \sigma^2) &\coloneqq D_{\mathrm{KL}}\Big(\mathcal{GP}(\mu^*, k^* \circ \sigma_*^2), \mathcal{GP}(\mu, k \circ \sigma^2)\Big) \\ &\approx D_{\mathrm{KL}}\bigg(\mathcal{N}(\boldsymbol{\mu}^*, \Sigma^*), \mathcal{N}\Big(\mu(\boldsymbol{x}), k \circ \sigma^2(\boldsymbol{x})\Big)\bigg) \\ &\approx D_{\mathrm{KL}}\bigg(\mathcal{N}(\tilde{\boldsymbol{\mu}}, \tilde{\Sigma}), \mathcal{N}\Big(\boldsymbol{\mu}, \Sigma\Big)\bigg), \end{aligned} \tag{5}$$

where $\boldsymbol{\mu} = \mu(\boldsymbol{x})$ and $\Sigma = k \circ \sigma^2(\boldsymbol{x})$. EKL in Eq. 5 measures the difference between the estimated multivariate Gaussian in Eq. 4 and a model $\mathcal{GP}(\mu, k \circ \sigma^2)$ evaluated on the inputs $\boldsymbol{x} = [x_j]_{j=1}^M$. Pre-training by minimizing EKL means aligning the model with an intermediate estimate of the ground truth GP on finite set of points. The KL divergence is well-defined for two non-degenerate Gaussians. We can insure the non-degeneracy of the model $\mathcal{GP}(\mu, k \circ \sigma^2)$ by constraining $k \circ \sigma^2$ to be positive definite (Eq. 3); i.e., covariance matrix $\Sigma = k \circ \sigma^2(\boldsymbol{x})$ is non-singular.

However, the estimated distribution $\mathcal{N}(\tilde{\boldsymbol{\mu}}, \tilde{\Sigma})$ is often degenerate since obtaining more datapoints on each task can be easier than defining and obtaining more training tasks. For example, our multi-task hyperparameter tuning benchmark in §7.1 has roughly 500 matching-input datapoints, but only 23 tasks (different model and dataset combinations). This is also true for other types of problems, such as robot skill learning (Wang et al., 2021), where the tasks are robot skills like scoop, pour etc, and the datapoints are score evaluations of control parameters. The tasks need to be defined and implemented carefully, but obtaining datapoints only requires repetitive experimentation. Hence, it is critical for us take into account that the estimated distribution $\mathcal{N}(\tilde{\boldsymbol{\mu}}, \tilde{\Sigma})$ can be degenerate.

We define D_{KL} in Eq. 5 as the KL divergence on the support of $\mathcal{N}(\tilde{\boldsymbol{\mu}}, \tilde{\Sigma})$. This is a generic definition of KL divergence that is inclusive of both degenerate and non-degenerate distributions.

To compute the support, we construct a matrix A such that $\tilde{\Sigma} = AA^\top$, $A \in \mathbb{R}^{M \times r}$ where $r \leq M$ is the rank of A and $\tilde{\Sigma}$. In practice, we use SVD to find a solution of matrix A by factorizing $\tilde{\Sigma}$ as $\tilde{\Sigma} = V \Lambda V^\top$ where $\Lambda \in \mathbb{R}^{r \times r}$ is a diagonal matrix with positive real-valued diagonal terms. We then set $A = V\sqrt{\Lambda}$ where $\sqrt{\Lambda}$ applies square root to all diagonal terms of Λ.

The pseudoinverse of matrix A is $A^+ = (A^\top A)^{-1} A^\top \in \mathbb{R}^{r \times M}$. We apply an affine transformation $\boldsymbol{v} = A^+(\boldsymbol{y} - \tilde{\boldsymbol{\mu}})$ to map any observation vector $\boldsymbol{y} \in \mathbb{R}^M$ to $\boldsymbol{v} \in \mathbb{R}^r$, which allows us to obtain marginal distributions by dropping irrelevant dimensions.

We can then compute D_{KL} as the KL divergence on the affine subspace of both distributions, $\mathcal{N}(\tilde{\boldsymbol{\mu}}, \tilde{\Sigma})$ and $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$, i.e.,

$$\begin{aligned} D_{\mathrm{KL}}\Big(\mathcal{N}(\tilde{\boldsymbol{\mu}}, \tilde{\Sigma}), \mathcal{N}(\boldsymbol{\mu}, \Sigma)\Big) &= D_{\mathrm{KL}}\Big(\mathcal{N}(0, I), \mathcal{N}(A^+(\boldsymbol{\mu} - \tilde{\boldsymbol{\mu}}), A^+ \Sigma (A^+)^\top)\Big) \\ &= \frac{1}{2}\left(\mathrm{tr}(\Sigma_p^{-1}) + (\boldsymbol{\mu}_p - \tilde{\boldsymbol{\mu}}_p)^\top \Sigma_p^{-1} (\boldsymbol{\mu}_p - \tilde{\boldsymbol{\mu}}_p) + \ln|\Sigma_p| - r\right), \end{aligned} \tag{6}$$

where $\boldsymbol{\mu}_p = A^+\boldsymbol{\mu}$, $\tilde{\boldsymbol{\mu}}_p = A^+\tilde{\boldsymbol{\mu}}$, $\Sigma_p = A^+\Sigma(A^+)^\top$.

Intuitively, the estimated Gaussian $\mathcal{N}(\tilde{\boldsymbol{\mu}}, \tilde{\Sigma})$ is only able to reflect information on a reduced dimensional subspace of the M-dimensional variable $\boldsymbol{y} \sim \mathcal{N}(\boldsymbol{\mu}^*, \Sigma^*)$ due to low rank. What we can

do is to make sure our model is aligned with the ground truth on that lower dimensional space, which can be achieved by minimizing Eq. 6.

Note the solution for matrix A is not unique. For any orthogonal matrix $B \in \mathbb{R}^{r\times r}$, $BB^\top = I$, we have $\tilde{\Sigma} = ABB^\top A^\top = AA^\top$, which means AB is also a solution. However, the KL divergence is invariant under parameter transformation. More specifically, the transformation is $\boldsymbol{v}' = (AB)^+(\boldsymbol{y} - \tilde{\boldsymbol{\mu}}) = U\boldsymbol{v}$ where $U = (B^\top A^\top AB)^{-1}B^\top A^\top A$. This is because

$$(AB)^+ = (B^\top A^\top AB)^{-1}B^\top A^\top = (B^\top A^\top AB)^{-1}B^\top A^\top AA^+.$$

Note that matrix U is non-singular[3]. Therefore, while matrix A is not unique, the KL divergence in Eq. 6 remains the same.

For the non-degenerate case of the estimated distribution $\mathcal{N}(\tilde{\boldsymbol{\mu}}, \tilde{\Sigma})$, we have $A^+ = A^{-1}$ and Eq. 6 recovers the standard definition of KL divergence between non-degenerate multivariate Gaussians:

$$\begin{aligned} D_{\text{KL}}\Big(\mathcal{N}(\tilde{\boldsymbol{\mu}}, \tilde{\Sigma}), \mathcal{N}(\boldsymbol{\mu}, \Sigma)\Big) &= D_{\text{KL}}\Big(\mathcal{N}(0, I), \mathcal{N}(A^{-1}(\boldsymbol{\mu} - \tilde{\boldsymbol{\mu}}), A^{-1}\Sigma(A^{-1})^\top)\Big) \\ &= \frac{1}{2}\left(\text{tr}(\Sigma^{-1}\tilde{\Sigma}) + (\boldsymbol{\mu} - \tilde{\boldsymbol{\mu}})^\top\Sigma^{-1}(\boldsymbol{\mu} - \tilde{\boldsymbol{\mu}}) + \ln\frac{|\Sigma|}{|\tilde{\Sigma}|} - M\right). \end{aligned} \tag{7}$$

More details on the derivation for EKL can be found in §B.

5.2.3 EXTENSIONS TO GENERIC CASES OF TRAINING DATA

In this section, we explore the case where our dataset D_N is not a "matching-input" dataset. That is, we have observations on different input locations for different training functions. Clearly, we cannot use the exact method for the case described in §5.2.2, but in fact, it is possible to heuristically transform our training data into a similar format as the "matching-input" dataset.

The core idea is to partition the input space into non-overlapping regions and merge the datapoints in each region to construct pseudo datapoints that align over training functions.

More formally, we define a partition strategy, $\psi : \mathfrak{X} \to \mathfrak{C}$, as a surjective function mapping inputs into a finite number of non-overlapping regions. We represent each region with a unique point in the original domain $\mathfrak{X}$; i.e., the range of partition strategy ψ is $\mathfrak{C} \subseteq \mathfrak{X}$, $\text{card}(\mathfrak{C}) < \infty$. For each training sub-dataset $D_{f_i} = \{(x_j^{(i)}, y_j^{(i)})\}_{j=1}^{M_i}$, $i \in [N]$, we can apply the partition to obtain a transformed dataset $\psi(D_N) \coloneqq \{\psi(D_{f_i})\}_{i=1}^N$ where $\psi(D_{f_i}) \coloneqq \{(\psi(x_j^{(i)}), y_j^{(i)})\}_{j=1}^{M_i}$. Given that $\psi(D_N)$ is a "matching-input" dataset, we can use the same method described in §5.2.2 to approximate the loss function in Eq. 1.

However, the performance of such an approach depends heavily on the partition strategy. In the literature, there exist many ways to obtain the partitions, e.g., Mondrian trees (Lakshminarayanan et al., 2016; Wang et al., 2018a), clustering methods (Omran et al., 2007), etc. More recently, Terenin et al. (2024) showed theoretical guarantees of partitioning data using covering trees; they also suggested summarizing the datapoints in each partition to a single datapoint with their mean observed values, while adjusting the weights of each summarized datapoint for efficient computation.

We leave the partitioning strategy as a future work. In §5.3.1, we show an alternative approximation for generic cases of training data.

3. This is because we can set $U^{-1} = (A^\top A)^{-1}BB^\top A^\top AB$ such that $UU^{-1} = U^{-1}U = I$.

5.2.4 MULTIPLE "MATCHING-INPUT" DATASETS

The "matching-input" dataset described in §5.2.1 requires observations on every training function. In practice, there could be multiple "matching-input" datasets, each of which has observations across a different set of training functions.

Let Q be the total number of these "matching-input" datasets, whose datapoints all exist in the training dataset D_N. We denote the q-th "matching-input" dataset as $\tilde{D}_q = \left\{ \{(x_j, y_j^{(i)})\}_{j \in \mathfrak{M}^{(q)}} \right\}_{i \in \mathfrak{N}^{(q)}}$, where $\mathfrak{N}^{(q)} \subseteq [N]$ is a set of indices for training functions that all have observations on a collection of inputs $\boldsymbol{x}^{(q)} = [x_j]_{j \in \mathfrak{M}^{(q)}}$, and $\mathfrak{M}^{(q)}$ is the set of indices for these inputs.

For each "matching-input" dataset $\tilde{D}_q$, we can use the same estimator in Eq. 4 to obtain estimated mean vector $\tilde{\boldsymbol{\mu}}^{(q)}$ and covariance $\tilde{\Sigma}^{(q)}$. We then approximate the loss function in Eq. 1 as follows.

$$\begin{aligned}
\mathcal{L}(\mu,\, k \circ \sigma^2) &\coloneqq D_{\mathrm{KL}}\Big(\mathcal{GP}(\mu^*, k^* \circ \sigma_*^2), \mathcal{GP}(\mu, k \circ \sigma^2)\Big) \\
&\approx \max_{q \in [Q]} D_{\mathrm{KL}}\Big(\mathcal{N}(\mu^*(\boldsymbol{x}^{(q)}), k^* \circ \sigma_*^2(\boldsymbol{x}^{(q)})), \mathcal{N}(\mu(\boldsymbol{x}^{(q)}), k \circ \sigma^2(\boldsymbol{x}^{(q)}))\Big) \\
&\approx \frac{1}{Q} \sum\nolimits_q^Q D_{\mathrm{KL}}\bigg(\mathcal{N}\Big(\tilde{\boldsymbol{\mu}}^{(q)}, \tilde{\Sigma}^{(q)}\Big), \mathcal{N}\Big(\boldsymbol{\mu}^{(q)}, \Sigma^{(q)}\Big)\bigg),
\end{aligned} \tag{8}$$

where $\boldsymbol{\mu} = \mu(\boldsymbol{x}^{(q)})$ and $\Sigma = k \circ \sigma^2(\boldsymbol{x}^{(q)})$. The last step of approximation with uniform weights is for computational convenience. For our experiments on the hyperparameter tuning benchmark described in §7.1, there are multiple "matching-input" datasets within our data, but each dataset may only have evaluations on a subset of the training functions. Hence we used Eq. 8 as the objective function in all of our experiments for EKL.

5.3 Negative Log Likelihood (NLL)

Alternative to directly computing EKL in Eq. 5, we can expand the KL divergence in the original loss function (Eq. 2) as follows,

$$\begin{aligned}
\mathcal{L}(\mu,\, k \circ \sigma^2) &= \sup\nolimits_{\boldsymbol{x} \subseteq \mathfrak{X}, \mathrm{card}(\boldsymbol{x}) < \infty} D_{\mathrm{KL}}\Big(\mathcal{N}(\mu^*(\boldsymbol{x}), k^* \circ \sigma_*^2(\boldsymbol{x})), \mathcal{N}(\mu(\boldsymbol{x}), k \circ \sigma^2(\boldsymbol{x}))\Big) \\
&= \sup\nolimits_{\boldsymbol{x} \subseteq \mathfrak{X}, \mathrm{card}(\boldsymbol{x}) < \infty} \mathbb{E}_{\boldsymbol{y} \sim \mathcal{N}(\mu^*(\boldsymbol{x}), k^* \circ \sigma_*^2(\boldsymbol{x}))} \log \frac{p(\boldsymbol{y} \mid \mu^*(\boldsymbol{x}), k^* \circ \sigma_*^2(\boldsymbol{x}))}{p(\boldsymbol{y} \mid \mu(\boldsymbol{x}), k \circ \sigma^2(\boldsymbol{x}))} \\
&= \sup\nolimits_{\boldsymbol{x} \subseteq \mathfrak{X}, \mathrm{card}(\boldsymbol{x}) < \infty} -\mathbb{E}_{\boldsymbol{y} \sim \mathcal{N}(\mu^*(\boldsymbol{x}), k^* \circ \sigma_*^2(\boldsymbol{x}))} \log p(\boldsymbol{y} \mid \mu(\boldsymbol{x}), k \circ \sigma^2(\boldsymbol{x})) + C^o
\end{aligned} \tag{9}$$

where C^o is a constant that does not depend on the model $\mathcal{GP}(\mu, k \circ \sigma^2)$. Without loss of generality, we omit this constant by setting $C^o = 0$ in the following approximations. In this section, we show that Eq. 9 can be approximated by the marginal log likelihoods of the training data. Eq. 9 is also closely related to cross-entropy losses typically used to pre-train deep learning models.

Case study. If the training dataset $D_N =$ is a "matching-input" dataset described in §5.2.1, we can approximate Eq. 9 by truncating the domain from $\mathfrak{X}$ to $\boldsymbol{x} = [x_j]_{j=1}^M$ and applying Monte Carlo estimation for the expectation; i.e.,

$$\mathcal{L}(\mu,\, k \circ \sigma^2) \approx -\frac{1}{N} \sum\nolimits_{i=1}^N \log p(\boldsymbol{y}_i \mid \mu(\boldsymbol{x}), k \circ \sigma^2(\boldsymbol{x})) = -\frac{1}{N} \sum\nolimits_{i=1}^N \log p(D_{f_i} \mid \mu, k \circ \sigma^2), \tag{10}$$

where $\boldsymbol{y}_i = [y_j^{(i)}]_{j=1}^M$ is the observed evaluations of training function f_i on inputs $\boldsymbol{x}$. Eq. 10 recovers the sum of the negative log marginal likelihoods of observations from *i.i.d.* training functions $f_1, \cdots, f_N$ and it is also an objective considered in multi-task learning (Caruana, 1997). A similar result can be shown for generic cases of training datasets that are not "matching-input".

5.3.1 NLL FOR GENERIC CASES OF TRAINING DATA

Now we investigate the general case of the training dataset $D_N = \{D_{f_i}\}_{i=1}^N$, where for each function, $D_{f_i} = \{(x_j^{(i)}, y_j^{(i)})\}_{j=1}^{M_i}$ has M_i datapoints and their input locations do not have to be the same as those of other functions. We illustrate the training dataset as follows.

f_1	$\cdots$	f_i	$\cdots$	f_N
$(x_1^{(1)}, y_1^{(1)})$	$\cdots$	$(x_1^{(i)}, y_1^{(i)})$	$\cdots$	$(x_1^{(N)}, y_1^{(N)})$
$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$
$(x_j^{(1)}, y_j^{(1)})$	$\cdots$	$(x_j^{(i)}, y_j^{(i)})$	$\cdots$	$(x_j^{(N)}, y_j^{(N)})$
$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$
$(x_{M_1}^{(1)}, y_{M_1}^{(1)})$	$\cdots$	$(x_{M_i}^{(i)}, y_{M_i}^{(i)})$	$\cdots$	$(x_{M_N}^{(N)}, y_{M_N}^{(N)})$

$$\boldsymbol{x}_i = \begin{bmatrix} x_1^{(i)} \\ \vdots \\ x_{M_i}^{(i)} \end{bmatrix} \qquad \boldsymbol{y}_i = \begin{bmatrix} y_1^{(i)} \\ \vdots \\ y_{M_i}^{(i)} \end{bmatrix}$$

Training dataset D_N includes observations on a series of finite sets of inputs, $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_N$. We further approximate the loss function in Eq. 9 by restricting the supremum to these sets of inputs:

$$\mathcal{L}(\mu,\, k \circ \sigma^2) \approx \max_{i \in [N]} -\mathbb{E}_{\boldsymbol{y} \sim \mathcal{N}(\mu^*(\boldsymbol{x}_i), k^* \circ \sigma_*^2(\boldsymbol{x}_i))} \log p(\boldsymbol{y} \mid \mu(\boldsymbol{x}_i), k \circ \sigma^2(\boldsymbol{x}_i)) \tag{11}$$

$$\approx -\sum_{i=1}^N w_i \log\; p(D_{f_i} \mid \mu, k \circ \sigma^2), \tag{12}$$

for some weight assignment w_i subject to $\sum_{i=1}^N w_i = 1$. The approximation step in Eq. 11 changes the supremum in Eq. 9 to a maximum over inputs $\boldsymbol{x}_i = [x_j^{(i)}]_{j=1}^{M_i}$ that exist in the training data. Eq. 12 uses one sample of $\boldsymbol{y} = \boldsymbol{y}_i$ to approximate the expectation and rewrites the maximum with a weight assignment. Sagawa et al. (2020) introduced a group distributionally robust optimization method that can be used to optimize the loss in Eq. 12. We set $w_i = \frac{1}{N}$ in this work and found in experiments that the uniform weighting is sufficient for GP pre-training.

Summary. The NLL approximation of the loss function on any training dataset $D_N = \{D_{f_i}\}$ is

$$\mathcal{L}(\mu, k \circ \sigma^2) \approx -\frac{1}{N} \sum_{i=1}^N \log p(D_{f_i} \mid \mu, k \circ \sigma^2). \tag{13}$$

For each training function f_i, the log marginal likelihood is

$$\log p(D_{f_i} \mid \mu, k, \sigma^2) = -\frac{1}{2} \left((\boldsymbol{y}_i - \boldsymbol{\mu}_i)^\top \Sigma_i^{-1} (\boldsymbol{y}_i - \boldsymbol{\mu}_i) + \log |\Sigma_i| + M_i \log 2\pi \right), \tag{14}$$

where $\boldsymbol{x}_i = [x_j^{(i)}]_{j=1}^{M_i}$, $\boldsymbol{y}_i = [y_j^{(i)}]_{j=1}^{M_i}$, $\boldsymbol{\mu}_i = \mu(\boldsymbol{x}_i)$ and $\Sigma_i = k \circ \sigma^2(\boldsymbol{x}_i)$.

5.4 Computational Complexity

In this section, we analyze the computational complexity of computing and optimizing the approximated loss functions, EKL in Eq. 5 and NLL in Eq. 13. We assume the loss functions are optimized

over fixed dimensional real-valued parameters $\theta \in \mathbb{R}^{d_\theta}$ ($d_\theta \ll \min(M, N)$). The optimization methods we analyze include gradient descent (GD) and stochastic gradient descent (SGD).

Recall that N is the number of training functions and let $M = \max_{i=1}^{N} M_i$ be the maximum number of datapoints observed on training functions. Table 1 summarizes our analyses.

		Time	Space
EKL (Eq. 5)	Overhead	$\mathcal{O}(M^2N)$	$\mathcal{O}(M^2)$
	Loss function	$\mathcal{O}(M^3)$	$\mathcal{O}(M^2)$
	GD	$\mathcal{O}(M^3K)$	$\mathcal{O}(M^2)$
	SGD	$\mathcal{O}(B^2MK)$	$\mathcal{O}(B^2)$
NLL (Eq. 13)	Loss function	$\mathcal{O}(M^3N)$	$\mathcal{O}(M^2)$
	Parallel	$\mathcal{O}(M^3)$	$\mathcal{O}(M^2N)$
	GD	$\mathcal{O}(M^3NK)$	$\mathcal{O}(M^2)$
	SGD	$\mathcal{O}(B^2MNK)$	$\mathcal{O}(B^2)$

Table 1: Time and space complexity. K is the number of optimization steps (or epochs in stochastic optimization). B is the mini-batch size of SGD over datapoints per training function.

EKL in Eq. 5 requires an overhead of estimating mean and covariance, which takes $\mathcal{O}(M^2N)$ for matrix multiplication. Once the mean and covariance are estimated, EKL has a time complexity of $\mathcal{O}(M^3)$ to compute and a space complexity of $\mathcal{O}(M^2)$.

NLL in Eq. 13 naturally decomposes into a sum of data likelihood terms on each sub-dataset D_{f_i}. The time complexity to compute Eq. 13 is $\mathcal{O}(M^3N)$, and the space complexity is $\mathcal{O}(M^2)$. If we have N processes computing each additive component of Eq. 13 in parallel, the time complexity can be reduced to $\mathcal{O}(M^3)$ while the space complexity becomes $\mathcal{O}(M^2N)$.

The main computational cost for EKL and NLL is solving linear systems (computing Σ^{-1} in Eq. 7 and Eq. 6; Σ_i^{-1} in Eq. 14). We can optionally use approximation methods for GPs in the literature to reduce the time complexity. For example, using V random features (Rahimi et al., 2007), the time complexity becomes $\mathcal{O}(V^3)$ instead of $\mathcal{O}(M^3)$.

Our method scales (at most) linearly with the number of tasks, N, in contrast to the cubic $\mathcal{O}(M^3N^3)$ scaling of multi-task or contextual GPs (Bonilla et al., 2007; Swersky et al., 2013; Bardenet et al., 2013; Poloczek et al., 2016; Yogatama and Mann, 2014). The only cubic cost is on the number of datapoints observed on each training function.

If we optimize the loss functions with SGD, the time complexity of computing the NLL objective (Eq. 13) can be reduced to $\mathcal{O}(B^2MN)$, where B is the mini-batch size of datapoints per training function. The space complexity reduces from $\mathcal{O}(M^2)$ in the original case to $\mathcal{O}(B^2)$ for the loss on mini-batches. However, SGD changes the optimization landscapes of both EKL and NLL.

In brief, both EKL in Eq. 5 and NLL in Eq. 13 enjoy low computational costs which scale at most linearly with the number of training functions. As a result, optimizing these approximated loss functions is not very expensive.

5.5 Relations between EKL and NLL

EKL and NLL are derived in different ways, but both approximate the same loss function in Eq. 1. In this section, we analyze the situations where EKL and NLL are equivalent (§5.5.1) and explain their important differences (§5.5.2) to provide deeper insights into the two objectives.

5.5.1 CONDITIONS FOR THE EQUIVALENCE OF EKL AND NLL

Claim. If the training dataset is a matching-input dataset described in §5.2.1, where $\boldsymbol{y}_i$ are *i.i.d.* samples from $\mathcal{N}(\boldsymbol{\mu}^*, \Sigma^*)$, and the estimated distribution $\mathcal{N}(\tilde{\boldsymbol{\mu}}, \tilde{\Sigma})$ in Eq. 4 is non-degenerate, EKL is equivalent to NLL up to a constant factor:

$$D_{\mathrm{KL}}\Big(\mathcal{N}(\tilde{\boldsymbol{\mu}}, \tilde{\Sigma}), \mathcal{N}(\boldsymbol{\mu}, \Sigma)\Big) \equiv -\frac{1}{N}\sum\nolimits_{i=1}^{N} \log p(\boldsymbol{y}_i \mid \boldsymbol{\mu}, \Sigma) + \frac{1}{N}\sum\nolimits_{i=1}^{N} \log p(\boldsymbol{y}_i \mid \tilde{\boldsymbol{\mu}}, \tilde{\Sigma}).$$

We prove this claim in §B.

5.5.2 DIFFERENCES BETWEEN EKL AND NLL

EKL naturally captures the correlations between function values by taking advantage of the matching inputs, while NLL is more flexible and naturally adapts to observations made at different input locations across training tasks. In different scenarios, one may choose either EKL or NLL. With NLL, each sub-dataset D_{f_i} may have different cardinality and arbitrary input locations. EKL works naturally with observations collected on the same inputs across training functions. While we can potentially use EKL for generic training datasets (§5.2.3), it generally involves more heuristics and manipulation of data than NLL.

§5.5.1 points out the interesting equivalence between EKL and NLL in special cases, but there remain important differences between EKL and NLL, including (1) quantitative evaluations, (2) implementation and data structures, (3) interpretation, and (4) extensions.

Quantitative evaluations. If the estimated distribution $\mathcal{N}(\tilde{\boldsymbol{\mu}}, \tilde{\Sigma})$ in Eq. 4 is degenerate, EKL and NLL can have different loss landscapes. Figure 3 shows the visualization of the quantitative differences between EKL and NLL for a toy problem, where the loss landscapes and $\arg\min$ locations of the two objectives are different.

Implementation and data structures. Regardless of whether the estimated distribution is degenerate, the implementation procedures of EKL and NLL are different. As discussed in §5.4, NLL can be N times more expensive to compute than EKL. Importantly, the implementation of EKL and NLL relies on different types of data structure.

It is more convenient for EKL to partition across inputs to use the [input, evaluations on all functions] structure $\{(x_j, [y_j^{(i)}]_{i=1}^N)\}_{j=1}^M$, and NLL to partition across functions to use the [input, evaluation per function] structure $\{\{(x_j, y_j^{(i)})\}_{j=1}^M\}_{i=1}^N$. During SGD, EKL first samples inputs and then computes the loss function value on those inputs, while it is more natural for NLL to sample functions and datapoints for each function.

In practice, there are multiple "matching-input" datasets (§5.2.4), and NLL would have to process the [input, evaluations on all functions] structure to combine all inputs if NLL were to use EKL's data structure. And for EKL to use NLL's [input, evaluation per function] structure, EKL would have to process the data to construct or find all matching inputs. More computational costs would incur in both of these scenarios.

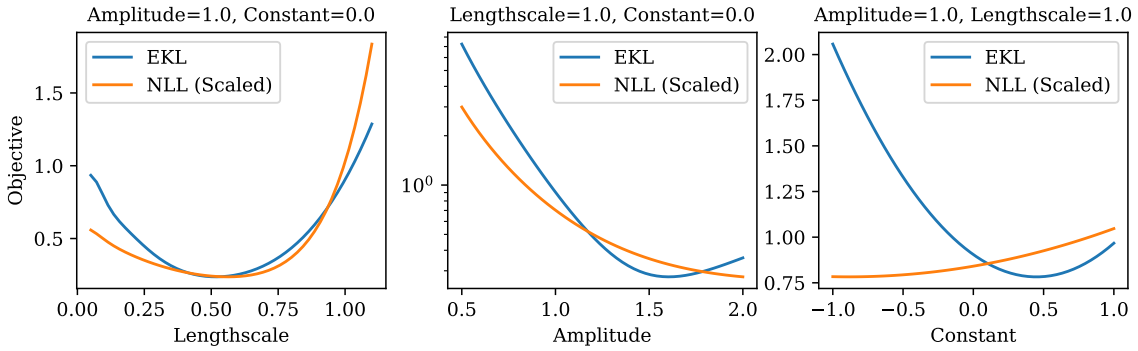

Figure 3: We define the a ground truth GP that has a zero mean and a Matern 5/2 kernel with amplitude 1.0 and lengthscale 1.0 on a 1-dimensional domain. We then sample 3 *i.i.d.* training functions from the GP and their evaluations on 5 inputs. Those 5 inputs are sampled *i.i.d.* from the standard normal distribution. The model is a GP with a constant mean function parameterized by a constant, and a squared exponential kernel parameterized by a lengthscale and an amplitude value. The figures visualize EKL and NLL (scaled by $\frac{\min EKL}{\min NLL}$ to allow consistency on scale) over each parameter with the other two fixed. In this setting, EKL and NLL have different landscapes and different $\arg\min$ locations.

Interpretation. Although the NLL loss function is more flexible, it can be difficult to interpret its correspondence to model quality, e.g., how high should the likelihood be for us to stop our search for a decent model? EKL in Eq. 5, on the other hand, is a divergence that is non-negative and equals 0 if and only if the two distributions are identical. One may choose to do early stopping or model selection based on how close EKL is to 0.

From the perspective of information theory (Cover and Thomas, 2006), we know that EKL in Eq. 5 can be interpreted as the average number of extra nats (or bits if we use 2 as the base of the logarithm instead of e) to encode the estimated multivariate Gaussian $\mathcal{N}(\tilde{\boldsymbol{\mu}}, \tilde{K})$ with a different model, compared to the the average number of nats to describe $\mathcal{N}(\tilde{\boldsymbol{\mu}}, \tilde{K})$. Comparatively, EKL in Eq. 5 is more interpretable than NLL in Eq. 13.

Extensions. EKL aims to measure the distance between two distributions: an estimate of the ground truth and the model. In this work, we use MLE as the estimate and the KL divergence as the "distance" for EKL, and establish the equivalence between solutions of NLL and EKL in special cases of data (Proposition 4). Extensions of EKL include using different estimators for the ground truth (e.g., the unbiased sample mean sample covaraince estimator) and adopting other distance measures (e.g., the Wasserstein distance or Rényi divergence). NLL, on the other hand, does not have these straightforward extensions.

6. HyperBO: Bayesian Optimization with Pre-trained Gaussian Process Priors

Bayesian optimization (BO) involves making decisions under uncertainty, trading off exploration and exploitation. In the previous sections, we developed pre-training methods to improve uncertainty estimates by leveraging existing data to specify better priors. The last piece of the puzzle is to

connect the pre-trained Gaussian process (GP) to BO methods. In this section, we present HyperBO, a general BO framework using the pre-trained GP as the prior.

As summarized in Algorithm 1, HyperBO is a simple wrapper over GP pre-training and classic BO steps for an unknown function f. We propose to fix the pre-trained GP in all BO steps (lines 4 to 8), so that we do not train the model and derive its posterior on the same set of datapoints D_f. Observations on function f are used only for posterior inference $p(f \mid D_f) = \mathcal{GP}(\hat{\mu}, \hat{k} \circ \hat{\sigma}^2 \mid D_f)$, but not any additional re-training steps (e.g., type II maximum likelihood) that modifies the mean and kernel functions of the GP.

Algorithm 1 HyperBO for optimizing unknown function f.

1: **function** HYPERBO(f, D_N)
2: $\quad \mathcal{GP}(\hat{\mu}, \hat{k} \circ \hat{\sigma}^2) \leftarrow$ PRE-TRAIN(D_N) ▷ Pre-train a GP on training dataset D_N (§5).
3: $\quad D_f \leftarrow \emptyset$
4: $\quad$ **for** $t = 1, \cdots, T$ **do**
5: $\quad\quad x_t \leftarrow \arg\max_{x \in \mathfrak{X}} \alpha\left(x; \mathcal{GP}(\hat{\mu}, \hat{k} \circ \hat{\sigma}^2 \mid D_f)\right)$ ▷ Optimize the acquisition function $\alpha(\cdot)$.
6: $\quad\quad y_t \leftarrow$ OBSERVE$(f(x_t))$ ▷ Collect noisy output of function f at input x_t.
7: $\quad\quad D_f \leftarrow D_f \cup \{(x_t, y_t)\}$
8: $\quad$ **end for**
9: $\quad$ **return** D_f
10: **end function**

HyperBO is a combination of an empirical Bayes[4] pre-training method and a fully Bayesian sequential decision making procedure. Once we obtain the estimated prior from pre-training, we treat it as the actual prior in the decision making module of HyperBO. Thus, we can circumvent the practical issues of unknown GP priors in BO. However, to HyperBO on an upper level, the pre-trained GP is not the ground truth GP prior. As we will see in §6.2 and §6.3, in some cases with few training functions, the difference between the ground truth and the pre-trained GP can lead to drastically different posterior predictions. This is harmful to BO. It is important to understand when this failure case can happen, so as to diagnose and avoid such scenarios.

6.1 Posterior Inference and Acquisition Strategies

In HyperBO, the acquisition function $\alpha(\cdot)$ (line 5 of Algorithm 1) can be any acquisition function that is fully defined by a Gaussian process (GP) posterior. However, it is important to keep in mind that the pre-trained model $\mathcal{GP}(\hat{\mu}, \hat{k} \circ \hat{\sigma}^2)$, is an approximation of the ground truth $\mathcal{GP}(\mu^*, k^* \circ \sigma_*^2)$. The relations between their corresponding acquisition function values are still unclear.

To understand the subtleties, we first compare the ground truth posterior and the pre-trained GP posterior. In the t-th iteration at line 5 of Algorithm 1, we have observations $D_f = \{(x_\tau, y_\tau)\}_{\tau=1}^{t-1}$. Let $\boldsymbol{x} = [x_\tau]_{\tau=1}^{t-1}$ and $\boldsymbol{y} = [y_\tau]_{\tau=1}^{t-1}$. The ground truth posterior is $\mathcal{GP}(\mu^*, k^* \circ \sigma_*^2 \mid D_f) := \mathcal{GP}(\mu_{t-1}^*, k_{t-1}^*)$, where

$$\mu_{t-1}^*(x) = \mu^*(x) + k^*(x, \boldsymbol{x})(k^* \circ \sigma_*^2(\boldsymbol{x}))^{-1}(\boldsymbol{y} - \mu^*(\boldsymbol{x})), \quad \forall x \in \mathfrak{X}, \tag{15}$$

$$k_{t-1}^*(x, x') = k^*(x, x') - k^*(x, \boldsymbol{x})(k^* \circ \sigma_*^2(\boldsymbol{x}))^{-1} k^*(\boldsymbol{x}, x'), \quad \forall x, x' \in \mathfrak{X}. \tag{16}$$

4. See §8.1 for a discussion on the Bayesian interpretations of HyperBO.

The pre-trained GP posterior is $\mathcal{GP}(\hat{\mu}, \hat{k} \circ \hat{\sigma}^2 \mid D_f) := \mathcal{GP}(\hat{\mu}_{t-1}, \hat{k}_{t-1})$, where

$$\hat{\mu}_{t-1}(x) := \hat{\mu}(x) + \hat{k}(x, \boldsymbol{x})(\hat{k} \circ \hat{\sigma}^2(\boldsymbol{x}))^{-1}(\boldsymbol{y} - \hat{\mu}(\boldsymbol{x})), \ \ \forall x \in \mathfrak{X}, \tag{17}$$

$$\hat{k}_{t-1}(x, x') := \hat{k}(x, x') - \hat{k}(x, \boldsymbol{x})(\hat{k} \circ \hat{\sigma}^2(\boldsymbol{x}))^{-1}\hat{k}(\boldsymbol{x}, x'), \ \ \forall x, x' \in \mathfrak{X}. \tag{18}$$

When optimizing the loss function (Eq. 1) or any of its approximates, we optimize over $\hat{k} \circ \hat{\sigma}^2$ together under the constraints that the kernel $\hat{k}$ is positive definite and the noise variance $\hat{\sigma}^2$ is positive. For the same $\hat{k} \circ \hat{\sigma}^2$, there can be multiple solutions for $\hat{k}$ and $\hat{\sigma}^2$, which further complicates the analyses. Even if we achieve the minimum of the loss function (Eq. 3), it is unclear if the following statement on the posterior is true:

$$\hat{\mu}_{t-1}, \hat{k}_{t-1} \ \stackrel{?}{=} \ \operatorname*{arg\,min}_{\mu, k} \ D_{\mathrm{KL}}\Big(\mathcal{GP}(\mu^*_{t-1}, k^*_{t-1}), \mathcal{GP}(\mu, k)\Big). \tag{19}$$

That is, if the pre-trained GP prior is close to the ground truth, is the pre-trained GP posterior also close to the ground truth posterior? We set this question aside for now and revisit in §6.2.1.

In HyperBO, since we have no access to the ground truth posterior, it is natural to construct acquisition strategies with the pre-trained GP posterior. Algorithmically, there is no constraint on what acquisition functions $\alpha(\cdot)$ should be used in HyperBO. For example, popular acquisition functions like GP-UCB (Kushner, 1962; Srinivas et al., 2010), EI (Šaltenis, 1971; Močkus, 1974) or PI (Kushner, 1962, 1964) are all directly applicable. The question remains whether an acquisition function based on the pre-trained GP reflects the strategy of the ground truth acquisition function, i.e.,

$$\operatorname*{arg\,max}_{x \in \mathfrak{X}} \ \alpha\left(x; \mathcal{GP}(\hat{\mu}, \hat{k} \circ \hat{\sigma}^2 \mid D_f)\right) \ \stackrel{?}{=} \ \operatorname*{arg\,max}_{x \in \mathfrak{X}} \ \alpha\left(x; \mathcal{GP}(\mu^*, k^* \circ \sigma^2_* \mid D_f)\right). \tag{20}$$

The open questions in Eq. 19 and Eq. 20 are the sufficient but not necessary conditions to obtain comparable regrets to the acquisition strategy with the ground truth GP prior.

We consider the following two acquisition functions for more analyses in §6.2. The acquisition functions are defined over the pre-trained GP posterior $\mathcal{GP}(\hat{\mu}, \hat{k} \circ \hat{\sigma}^2 \mid D_f)$ in Eq. 17 and Eq. 18.

- A variant of GP-UCB (Srinivas et al., 2010) with explore-exploit trade-off parameter β:

$$\alpha^{\mathrm{UCB}}_{t-1}(x) = \hat{\mu}_{t-1}(x) + \beta\sqrt{\hat{k}_{t-1} \circ \hat{\sigma}^2(x)}. \tag{21}$$

- A variant of the max-value version of PI (Wang and Jegelka, 2017; Kushner, 1964) with $\hat{f}^*$ as an estimate of the max-value of the test function f:

$$\alpha^{\mathrm{PI}}_{t-1}(x) = \frac{\hat{\mu}_{t-1}(x) - \hat{f}^*}{\sqrt{\hat{k}_{t-1} \circ \hat{\sigma}^2(x)}}. \tag{22}$$

As shown by Wang and Jegelka (2017), GP-UCB and PI are closely related to entropy search based methods (Hennig and Schuler, 2012; Hernández-Lobato et al., 2014), and the max-value of test function f can be estimated from the posterior on f.

Next, we conduct a case study on finite input domain $\mathfrak{X}$ and provide theoretical insights to understand asymptotic behaviours of the pre-trained GP posterior predictions in §6.2.1. As part of the verification for HyperBO, we present its regret bounds (§6.2.2) with unknown ground truth GP priors. We explain the intuitions of these theoretical analyses with synthetic examples in §6.3.

6.2 Case Study: Functions with Finite Domains

We study the theoretical aspects of HyperBO where the cardinality of the input domain, $|\mathfrak{X}|$, is finite. We can then consider the perfect case where we observe all values of the training functions $f_1, \cdots, f_N$. That is, we further strengthen the assumption in §5.2.1 to be "observing training functions on all inputs in the finite domain".

Assumption 3. *The domain $\mathfrak{X} = \{x_j\}_{j=1}^M$ contains a finite number of inputs. The training dataset is $D_N = \{D_{f_i}\}_{i=1}^N$ where $D_{f_i} = \{(x_j, y_j^{(i)})\}_{j=1}^M$. Assume $N > M$ and the estimated covariance matrix in Eq. 4 is full rank.*

Now, we can show the pre-trained $\mathcal{GP}(\hat{\mu}, \hat{k} \circ \hat{\sigma}^2)$ has a closed form solution.

Proposition 4. *Let $\boldsymbol{x} = [x_j]_{j=1}^M \in \mathbb{R}^{M \times d}, Y = [y_j^{(i)}]_{j \in [M], i \in [N]} \in \mathbb{R}^{M \times N}$. Given Assumption 1, 2 and 3, we have*

$$\hat{\mu}(\boldsymbol{x}) = \frac{1}{N} Y 1_N \quad \textit{and} \quad \hat{k} \circ \hat{\sigma}^2(\boldsymbol{x}) = \frac{1}{N}(Y - \hat{\mu}(\boldsymbol{x}) 1_N^\top)(Y - \hat{\mu}(\boldsymbol{x}) 1_N^\top)^\top \tag{23}$$

such that $\hat{\mu}, \hat{k} \circ \hat{\sigma}^2 = \operatorname{arg\,min}_{\mu, k \circ \sigma^2} \mathcal{L}^{EKL}(\mu, k \circ \sigma^2) = \operatorname{arg\,min}_{\mu, k \circ \sigma^2} \mathcal{L}^{NLL}(\mu, k \circ \sigma^2)$, where the loss function $\mathcal{L}^{EKL}$ is defined in Eq. 5 and $\mathcal{L}^{NLL}$ is defined in Eq. 13.

Proposition 4 is not difficult to show. The EKL loss function is a KL divergence between two multivariate Gaussian distributions. EKL is non-negative and reaches its minimum value 0 if the two distributions are the same. Eq. 23 ensures that the pre-trained model $\mathcal{N}(\hat{\mu}(\boldsymbol{x}), \hat{k} \circ \hat{\sigma}^2(\boldsymbol{x}))$ and the estimated model $\mathcal{N}(\tilde{\boldsymbol{\mu}}, \tilde{\Sigma})$ from Eq. 4 are the same, so that the pre-trained model minimizes EKL. The solution to the NLL objective becomes the maximum likelihood estimate, which is the same as Eq. 4. To show that there exists functions $\hat{\mu}$ and $\hat{k} \circ \hat{\sigma}^2$ that satisfy Eq. 23, we can construct a simple memory based model. The model stores each element of vector $\tilde{\boldsymbol{\mu}}$ and matrix $\tilde{\Sigma}$ from Eq. 4. When making a prediction at any input or pairs of inputs, the model simply retrieves the corresponding mean or covariance values saved in the model.

Recall that one of the open questions we had in §6.1 is the proximity between pre-trained GP posteriors and ground truth GP posteriors. Assumption 3 and Proposition 4 enable us to bound pre-trained GP posterior predictions with ground truth posterior predictions. The following analyses in §6.2 all assume Assumptions 1, 2 and 3.

6.2.1 BOUNDING PRE-TRAINED GP POSTERIOR PREDICTIONS

As discussed in §6.1, we are interested in the relation between the pre-trained GP posterior $\mathcal{GP}(\hat{\mu}, \hat{k} \circ \hat{\sigma}^2 \mid D_f)$ and the ground truth posterior $\mathcal{GP}(\mu^*, k^* \circ \sigma_*^2 \mid D_f)$ for each iteration of HyperBO in Algorithm 1. Theorem 5 shows that the pre-trained GP posterior mean and variance are bounded by the ground truth posterior mean and variance.

Theorem 5. *Assume $N > T + 1$ and $T \geq t \geq 1$. At the t-th iteration of HyperBO in Algorithm 1, for any input $x \in \mathfrak{X}$, we have*

$$\mathbb{E}[\hat{\mu}_{t-1}(x)] = \mu_{t-1}^*(x) \quad \textit{and} \quad \mathbb{E}[\hat{k}_{t-1} \circ \hat{\sigma}^2(x)] = \frac{N - t}{N} k_{t-1}^* \circ \sigma_*^2(x).$$

With probability at least $1-\delta$,

$$|\hat{\mu}_{t-1}(x)-\mu^*_{t-1}(x)|^2 < a^2 k^*_{t-1}\circ\sigma^2_*(x) \text{ and } 1-2\sqrt{b} < \frac{N\hat{k}_{t-1}\circ\hat{\sigma}^2(x)}{(N-t)k^*_{t-1}\circ\sigma^2_*(x)} < 1+2\sqrt{b}+2b,$$

where $a^2 = \frac{4\left(t+2\sqrt{(t-1)\log\frac{4}{\delta}}+2\log\frac{4}{\delta}-2/N\right)}{(N-t-1)\delta}, b = \frac{1}{N-t}\log\frac{4}{\delta}$.

The proof can be found in §C.1. Theorem 5 conveys an interesting finding: as the iteration t increases, both bounds become looser and the pre-trained GP posterior variance gradually becomes more biased. As a remedy, we can readjust the scale of the pre-trained GP posterior variance by $\frac{N}{N-t}$ to match the ground truth in expectation.

Perhaps counter-intuitively, Theorem 5 means that we need to have less confidence in the pre-trained GP as we observe more data from a test function. Yet the epistemic uncertainty predictions from the pre-trained GP become smaller with more observations. We can understand this result as a rivalry between the posterior conditioned on a pre-trained GP prior and the uncertainty that comes with the pre-training approximation of the ground truth. Theorem 5 also implies that BO with pre-trained GPs is more reliable when there are more training functions and datapoints per training function than the number of BO iterations.

6.2.2 THEORETICAL ANALYSES ON REGRET BOUNDS

The other open question in §6.1 is essentially about how acquisition functions with pre-trained GP posteriors impact the performance of BO. We show a near-zero regret bound for HyperBO under Assumption 1, 2 and 3, i.e., unknown ground truth models and full observations on training functions in finite input domains. The setup of HyperBO under Assumption 3 is equivalent to the finite-arm bandit problem where the values of the arms are distributed according to a multivariate Gaussian.

Theorem 6. *Let* $N > T + 4\log\frac{12}{\delta}$, $\delta\in(0,1)$. *With probability at least* $1-\delta$, *the simple regret in* T *iterations of Algorithm 1 with special cases of either GP-UCB in Eq. 21 or PI in Eq. 22 satisfies*

$$R_T < O\left(\sqrt{\frac{T}{N-T-1}}+\sqrt{\log\frac{1}{\delta}}\right)O(\sqrt{\rho_T/T}+\sigma_*), \tag{24}$$

where $\rho_T = \max_{A\subset\mathfrak{X},|A|=T}\frac{1}{2}\log|\boldsymbol{I}+\sigma_*^{-2}k^*(A)|$.

We describe details of the proof in Appendix C.2. In Appendix C.3, we also provide a regret bound defined by the pre-trained GP instead of the ground truth. Theorem 6 shows that the regret bound always has a linear dependency on the observation noise σ. This is expected because in practice, we select the best observation rather than best function value (before observing a noisy version of it) to compute the simple regret. Another reason is that we learn the noise parameter σ jointly with the kernel, as shown in Eq. 1, and when computing acquisition functions (Eq. 21 or Eq. 22), the noise parameter σ is always included in the predicted variance.

Intuitively, the more sub-datasets we have in the training dataset, the larger N is, the better we are able to estimate the ground truth GP model, and the closer the regret bound is to the case where the ground truth GP model is assumed known. Interestingly, the number of BO iterations, T, makes the regret smaller in the second term but larger in the first term in Eq. 24. Usually as we get more

observations, we get more information about the maximizer, and we are able to optimize the function better. However, as we get more observations on the new function, the pre-trained GP posterior predictions have more freedom to deviate from the ground truth, as shown by Theorem 5. Hence, we get less and less confident about our predictions, which eventually leads to a looser regret bound.

It is tempting to prove similar bounds for more general settings where inputs are not the same across training functions or the input domain is continuous. Though the only prerequisite is to show that the difference between the pre-trained mean/kernel and the ground truth mean/kernel is small, this prerequisite is as difficult as showing we can find a model that has bounded generalization error across the entire continuous input space of an arbitrary function. We leave the regret bound for general settings as an open question.

6.3 Understanding HyperBO with Synthetic Examples

We here ground our theory on simple synthetic setups to more intuitively understand HyperBO. Note that the critical component in BO is posterior inference, since all decision making relies on the posterior. Hence we focus on the posterior aspects of HyperBO in this section.

Our analyses also rely on the notation of functions. Mathematically, function representations can involve infinite-dimensional vectors or finite parameterization with specific modeling choices. But from a computer science perspective, on a compact domain, we can represent a function with a finite number of datapoints without specifying parameterized models. This is because in a computer, numbers are represented by floats, and floats use finite bits. For example, for optimization trajectories, the points have to move at least a delta (e.g., 6.1×10^{-5} for float16) at a time. So it is sufficient for us to consider a function fully represented by finite evaluations.

In the following, §6.3.1 considers small function domains with varying numbers of training functions, and provides intuitions on Theorem 5. §6.3.2 partly invalidates Theorem 5 in settings with stationary kernels, few training functions, but more training datapoints per function.

6.3.1 SIMPLE FUNCTION DOMAINS

Here, we assume the ground truth model is a GP with zero noise variance, zero mean and squared exponential kernel on a 1-dimensional domain. The lengthscale parameter and signal variance of the GP are both 1. That is,

$$\mu^*(x) = 0 \quad \text{and} \quad k^* \circ \sigma_*^2(x, x') = e^{-(x-x')^2/2}, \forall x, x' \in \mathfrak{X}.$$

We consider the simple setup from §6.2. We set the domain to be $\mathfrak{X} = \{1, 2, 3, 4, 5\}$ and collect a training dataset $D_n = \{D_{f_i}\}_{i=1}^N$ by sampling from the GP, which is equivalent to a multivariate normal distribution. The top left of Figure 4 shows the samples, where each sub-dataset D_{f_i} is illustrated with the same color.

In Figure 4, we illustrate the prior and posterior predictions from the ground truth and three modeling choices: the MLE estimate in blue, the unbiased estimate in red and the pre-trained GP in green. The MLE estimate uses Eq. 4. The unbiased estimator applies a rescaling factor, $\frac{N}{N-t}$, to MLE, and obtain unbiased estimates, where t is the number of observations on test function f. If $t = 1$, the unbiased estimate is the sample mean and covariance. The pre-trained GP optimizes the EKL loss function (Eq. 7) over unknown parameters of a squared exponential kernel.

With no observations, the prior estimates (top middle plot) all look aligned with the ground truth. However, as we increase the number of observations from 1 (top right) to 4 (bottom right), posterior

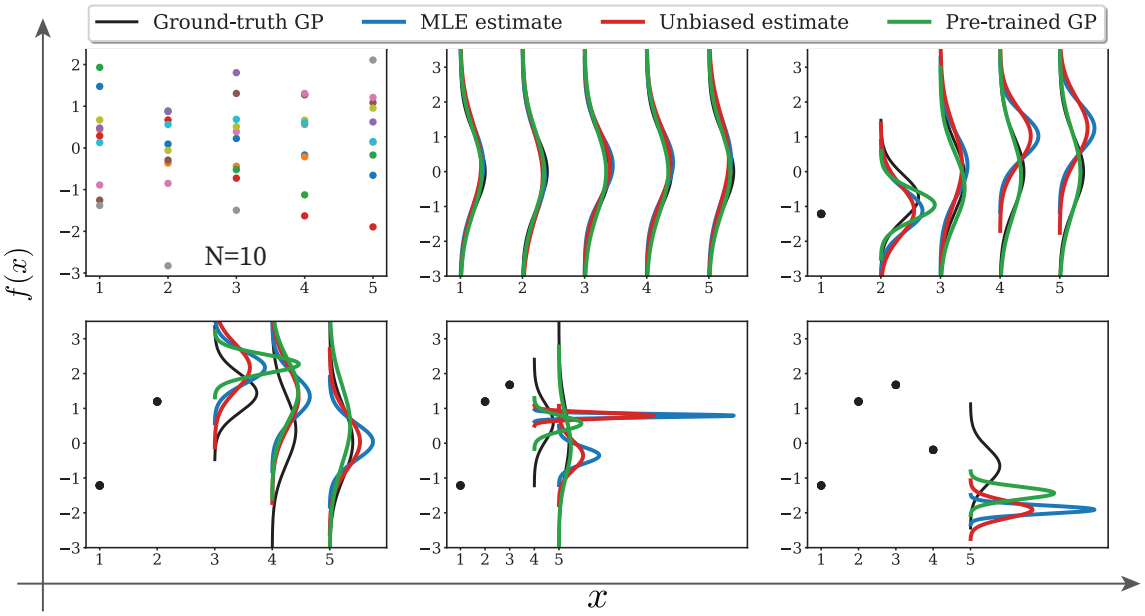

Figure 4: Top left shows 10 training functions, each with one color, sampled from a ground truth GP (a multivariate Gaussian) on a finite domain $\mathfrak{X} = \{1, 2, 3, 4, 5\}$. The top middle plot shows probability densities of the marginal Gaussian distribution for each function value evaluated with the ground truth, MLE estimate, unbiased estimate and pre-trained GP. The following plots show their conditional distributions (i.e., posteriors). With increased observations (black dots), estimate-based posterior predictions become less accurate despite close estimations of the prior.

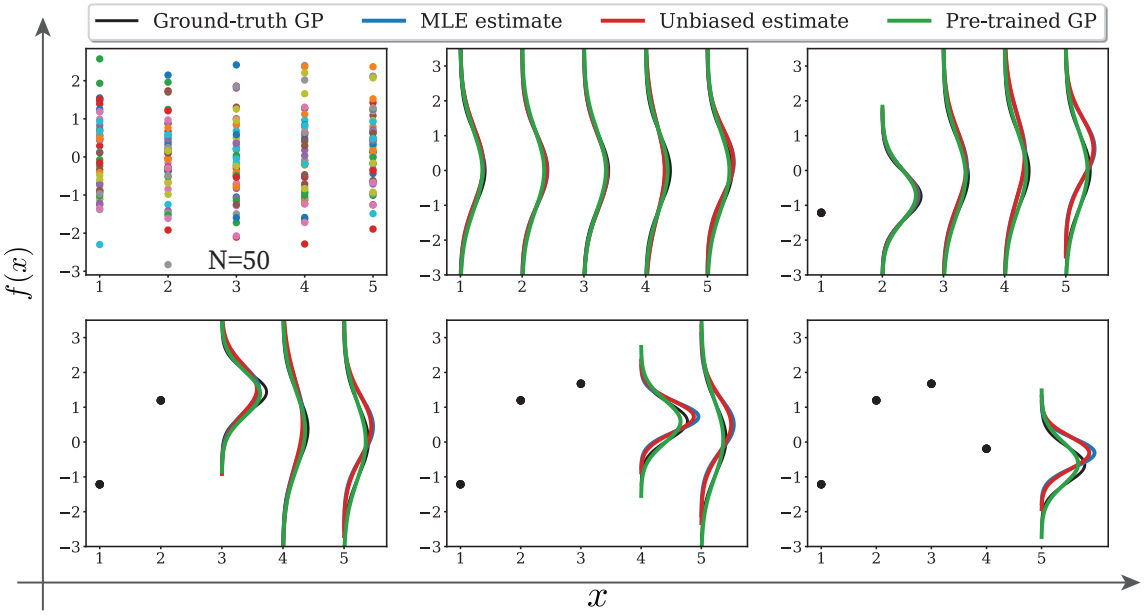

Figure 5: We used the same setup as Figure 4, except that the size of the training dataset is $N = 50$. With more training functions, we obtain more accurate pre-trained GP posterior predictions.

predictions based on the estimates deviate more and more from the ground truth posteriors. This confirms the results from Theorem 5, which says the bounds on estimates rely on $N - t$, so the more observations, the less accurate the estimated posterior predictions become with respect to the ground truth. Note that by applying rescaling in the unbiased posterior estimate, we can avoid the overly confident posterior predictions from MLE.

In Figure 5, we increased the size of the training dataset from 10 in Figure 4 to 50. More training functions allow us to obtain more accurate posterior predictions based on all 3 kinds of modeling choices. Rescaling also becomes less important since $t \ll N$. Interestingly, for both Figure 4 and Figure 5, the pre-trained GP produces more accurate posterior predictions compared to either MLE or unbiased estimates.

6.3.2 FUNCTION AUGMENTATION BY ADDING MORE TRAINING DATAPOINTS

Note that the above analyses for pre-training with few training functions may not hold in some practical settings. If the unknown ground truth kernel is stationary and there are enough observations per training function, as shown by Bachoc (2021), we might still be able to obtain a good pre-trained GP. For example, in Figure 1, there are only 3 training functions, but each training function has 200 datapoints covering the input domain. Figure 6 shows prior and posterior predictions made by the ground truth, pre-trained and single-task GPs.

The ground truth GP used a constant noise variance, a Matérn52 kernel and a linear MLP mean function with 3 hidden layers. We used a constant noise variance, a squared exponential kernel and the same setup of a linear MLP mean function for the pre-trained GP. The single-task GP used constant mean and squared exponential kernel. The parameters of mean and kernel functions are usually referred to as GP hyperparameters (Rasmussen and Williams, 2006). We performed type-II maximum likelihood to obtain GP hyperparameters of the single-task GP on 3 observations from the test function. Table 2 shows 4 metrics: (1) the losses defined by NLL in Eq. 13, (2) EKL in Eq. 6, both over the training functions, (3) NLL(f): the NLL on the total 200 datapoint of the test function f, and (4) NLL(D_f): the NLL on the 3 observations on test function f. Single-task GP obtains the lowest NLL(D_f) but all other 3 metrics are higher than other models, implying overfitting.

	NLL	EKL	NLL(f)	NLL(D_f)
ground truth GP	-437.1	3.2	-434.5	3.3
Pre-trained GP	-244.4	3.6	-271.5	2.8
Single-task GP	181.3	114.1	246.8	1.6

Table 2: The NLL (Eq. 13) and EKL (Eq. 6) loss function values on training functions, together with NLL(f): the NLL on 200 function evaluations on test function f, and NLL(D_f): the NLL on the 3 observations on test function f. While the single-task GP obtained the lowest NLL on the 3 observations, it is not a good model for the underlying test function as reflected by the high NLL(f) values. On the contrary, the pre-trained GP provided a better fit for test function f.

Why does a pre-trained GP still work on such a small set of training functions? We can understand this through the SGD setup, where in every training step, a batch of datapoints are sampled from each training sub-dataset. Given that we have a stationary kernel, each batch can be viewed as points on a different training function. Thus, we can effectively augment the set of training functions, so

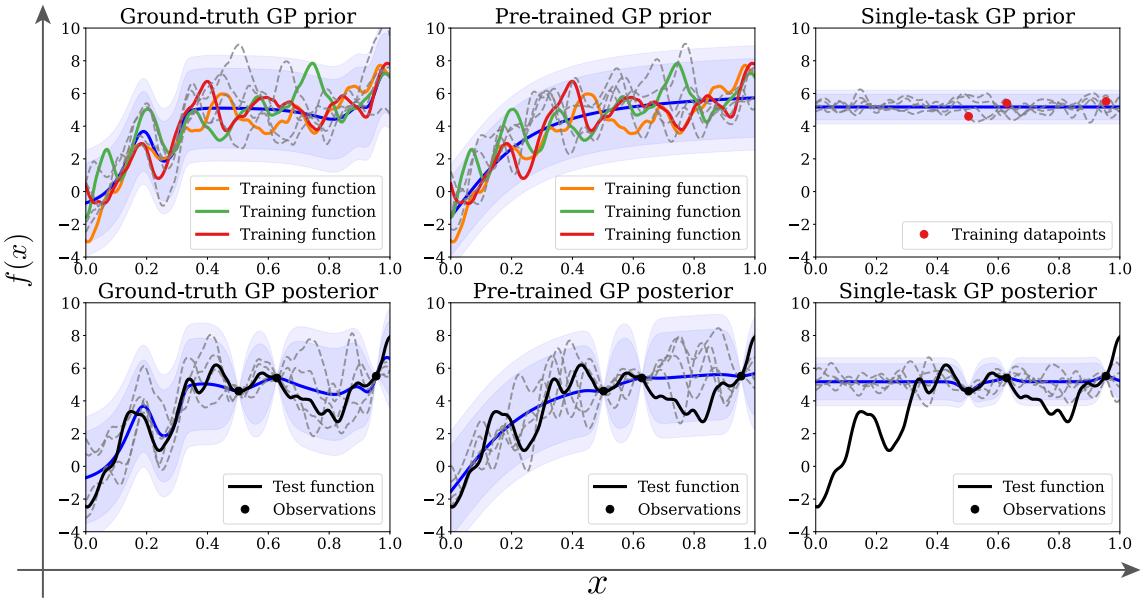

Figure 6: Comparing priors and posteriors based on the ground truth GP, pre-trained GP trained on 3 functions (colored lines), and single-task GP optimized with type-II maximum likelihood on 3 observations (black dots). GP mean functions are illustrated in blue and the shaded areas are 95% and 99% confidence intervals. The grey dotted lines are random samples from the GP in each figure. The confidence intervals from the single-task GP posterior failed to cover the test function (the black line), though providing a good fit for the observations. The pre-trained GP did not share the same kernel type as the ground truth. Despite having only 3 training functions, the overall trend of the pre-trained GP posterior matched the ground truth posterior and would be useful for BO.

that we won't be constrained by Theorem 5. Concurrently, Fan et al. (2024) showed analyses with more precise theoretical and empirical analyses.

7. Experiments

In this section, we evaluate the performance of HyperBO on a variety of real-world hyperparameter tuning tasks. In total, we performed experiments on 17 search spaces. Each search space has multiple tasks, and the tasks correspond to black-box functions (e.g., for evaluating validation error rates) over the same set of hyperparameters of a system. For every search space, the tasks are divided into training tasks and test tasks. The datapoints from the training tasks compose a training dataset, which we can use to pre-train a Gaussian process (GP). The performance of Bayesian optimization (BO) is evaluated on the test tasks.

Among those search spaces, the first one corresponds to the problem of hyperparameters tuning for the optimizer of modern deep learning models, and the goal is to obtain a tuning strategy that can generalize over different combinations of model architectures (e.g., ResNet50 from He et al., 2016), datasets (e.g., ImageNet from Russakovsky et al., 2015) and hardware settings (which determine batch sizes). We created a new benchmark, PD1 (§7.1), for this challenging tuning problem of deep learning. PD1 is the focus of our experiments given its relevance to present-day large scale

deep learning applications. The goals of our experiments on PD1 are to investigate the performance of HyperBO and understand its properties. In §7.4, we present the simulated offline BO and GP regression results on PD1 with detailed studies on the effects of the number of training datapoints and the number of training tasks. And in §7.5, we show the online tuning results for tasks in PD1 to demonstrate the performance of HyperBO when it is deployed online as part of a real world hyperparameter tuning service.

The rest of the search spaces belong to HPO-B (Pineda-Arango et al., 2021), a hyperparameter tuning benchmark for relatively small scale but more classic machine learning models such as decision trees and SVM. To further showcase the performance and robustness of HyperBO, we present results for both the original HPO-B and representative variants of HPO-B with reduced training data and the "negative transfer" effects (Rothfuss et al., 2021) in §7.6.

Our JAX-based (Bradbury et al., 2018) implementation of HyperBO can be found at `https://github.com/google-research/hyperbo`, which was used for all of our experiments. To accommodate needs for more modular use cases, we also provide a Flax (Heek et al., 2020) and TensorFlow-Probability (Dillon et al., 2017) based implementation for GP pre-training at `https://github.com/google-research/gpax`.

In the following, we introduce PD1 in §7.1, describe compared methods in §7.2 and evaluation metrics in §7.3, and present the aforementioned results with analyses on PD1 and HPO-B. We give a summary of the experiments in §7.7.

7.1 PD1: A New Hyperparameter Tuning Benchmark for Optimizing Deep Learning Models

To collect our hyperparameter tuning dataset, the PD1 Neural Net Tuning Dataset, we defined a set of 24 neural network tuning tasks and a single, broad search space for Nesterov momentum (Nesterov, 1983; Sutskever et al., 2013). Each task is defined by a task dataset (e.g., ImageNet from Russakovsky et al., 2015), a specific neural network model (e.g., ResNet50 from He et al., 2016), and a batch size (which is determined by the hardware).

To reduce ambiguity, we distinguish between datasets that individual neural networks are trained on and the dataset we collected that includes optimizer hyperparameter points with their validation errors (and other metrics). We will call the former, e.g., MNIST (LeCun et al., 2010) and CIFAR10 (Krizhevsky, 2009), task datasets and call the latter the tuning dataset. The tuning dataset is what we described as dataset D_N in §4.

Table 3 shows all the tasks that we consider in the tuning dataset. We used an existing code base (Gilmer et al., 2021) for neural network model training. The dataset used roughly 12,000 machine-days of computation on TPUv4i (Jouppi et al., 2021) for approximately 50,000 hyperparameter evaluations. Depending on the task, the runtime of each hyperparameter evaluation may vary from minutes to days.

For each task, we trained the model on the task dataset repeatedly using Nesterov momentum (Nesterov, 1983; Sutskever et al., 2013), with the task's minibatch size, with different hyperparameter settings drawn from the 4-dimensional search space detailed in Table 4. We tuned the base learning rate, η, on a log scale, the momentum, β, with $1-\beta$ on a log scale, and the polynomial learning rate decay schedule power p and decay steps fraction λ. We used a polynomial decay schedule with the following form:

$$\eta_\tau = \frac{\eta}{1000} + \left(\eta - \frac{\eta}{1000}\right)\left(1 - \frac{\min(\tau, \lambda\mathcal{T})}{\lambda\mathcal{T}}\right)^p, \tag{25}$$

where τ is the training step and $\mathcal{T}$ is the total number of training steps for the task.

We collected two types of data: matched and unmatched data. Matched data used the same set of uniformly-sampled hyperparameter points (i.e., "matching-input" in §5.2.1) across all tasks and unmatched data sampled new points for each task. All other training pipeline hyperparameters were fixed to hand-selected, task-specific default values. All of our tasks are classification problems, so they all used the same training loss, although occasionally task-specific regularization terms were added. For each trial (training run for a single hyperparameter point), we recorded validation error (both cross entropy error and misclassification rate). In many cases, poor optimizer hyperparameter choices can cause training to diverge. We detected divergent training when the training cost became NaN and then marked the trial but did not discard it. Please download the dataset (`http://storage.googleapis.com/gresearch/pint/pd1.tar.gz`) and see its descriptions for additional details about the tasks and training procedure. The different tuning tasks vary in difficulty and numbers of datapoints, but generally there are roughly 500 matched datapoints and 1500 unmatched datapoints per tuning task. Among the 500 matched datapoints, 242 datapoints correspond to training runs that did not diverge. For unmatched data only, we attempted to generate roughly similar numbers of non-divergent points across tasks, so tasks with a higher probability of sampling a hyperparameter point that causes training to diverge will tend to have more trials.

The ImageNet ResNet50 1024 task only has 100 hyperparameter points because we abandoned it when scaling up data collection in order to save compute resources. It is used in training, but not evaluation. In total, we have 23 test tasks in PD1. For each test task, we used subsets of the other 23 tasks (including ImageNet ResNet50 1024) to compose training datasets.

Task Dataset	Model	Batch Sizes
CIFAR10	Wide ResNet	{256, 2048}
CIFAR100	Wide ResNet	{256, 2048}
Fashion MNIST	Max pool CNN ReLU	{256, 2048}
Fashion MNIST	Max pool CNN tanh	{256, 2048}
Fashion MNIST	Simple CNN	{256, 2048}
ImageNet	ResNet50	{512, 1024, 2048}
LM1B	Transformer	{2048}
MNIST	Max pool CNN relu	{256, 2048}
MNIST	Max pool CNN tanh	{256, 2048}
MNIST	Simple CNN	{256, 2048}
SVHN (no extra)	Wide ResNet	{256, 1024}
WMT15 German-English	xformer	{64}
UniRef50	Transformer	{128}

Table 3: Tasks in PD1.

Hyperparameter	Range	Warping
η	$[10^{-5}, 10]$	Log
p	$[0.1, 2.0]$	Linear
$1-\beta$	$[10^{-3}, 1.0]$	Log
λ	$[0.01, 0.99]$	Linear

Table 4: 4-dimensional input search space of PD1 (see Eq.25).

For our experiments on PD1, we also used output warping in addition to input warping described in Table 4. The validation error rate outputs are warped as $r \leftarrow -\log(r + 10^{-10})$ for all methods unless otherwise mentioned, so that we have a maximization problem for tuning.

7.2 Description of All Compared Methods

We compared two sets of methods, one was a collection of meta BO methods, including HyperBO, and the other was for single task BO methods that did not make use of multi-task training data.

Our method HyperBO has several variants including using different acquisition functions and different objectives. In our experiments, unless otherwise mentioned, we used a thresholded probability

of improvement (PI) as the acquisition function. We set PI[5] in line 5 of Algorithm 1 as

$$\alpha\left(x; \mathcal{GP}(\hat{\mu}, \hat{k} \circ \hat{\sigma}^2 \mid D_f = \{(x_\tau, y_\tau)_{\tau=1}^{t-1}\})\right) = \frac{\hat{\mu}_{t-1}(x) - (\max_{\tau \in [t-1]} y_\tau + 0.1)}{\sqrt{\hat{k}_{t-1} \circ \hat{\sigma}^2(x)}}.$$

To optimize the loss functions (§5), while there exist methods to search for functional structures (Kemp and Tenenbaum, 2008; Malkomes and Garnett, 2018), we opted for fixed but complex and expressive structures of mean and kernel functions. We then optimize the objective directly via gradient based methods. Alternatively, one can perform cross-validation on a held-out validation dataset (Wistuba and Grabocka, 2021), which we skipped for simplicity and speed.

We included the following HyperBO variants in our experiments.

- H-NLL: HyperBO with a GP pre-trained via the NLL objective (Eq. 13). We used a 2-hidden-layer neural network of size $(32, 32)$ as mean function and an anisotropic Matérn52 covariance on the last feature layer of the mean function as kernel.

  We used tanh activation for the neural network. The neural network gives us a mapping from inputs to embeddings. Let the mapping be $\phi : \mathfrak{X} \mapsto \mathbb{R}^{d_{\text{emb}}}$, a Matérn covariance function be $k_{\text{Matérn}} : \mathbb{R}^{d_{\text{emb}}} \times \mathbb{R}^{d_{\text{emb}}} \mapsto \mathbb{R}$, and a weight matrix be $W \in \mathbb{R}^{1 \times d_{\text{emb}}}$. The mean function used by H-NLL is $\mu(x) = W\phi(x)$ for any $x \in \mathfrak{X}$. The kernel function used by H-NLL is $k(x, x') = k_{\text{Matérn}}(\phi(x), \phi(x'))$ for any $x, x' \in \mathfrak{X}$.

  As part of the GP model, we also used softplus warping functions for kernel and likelihood parameters. The NLL objective was optimized with the Adam optimizer (Kingma and Ba, 2015) implemented in Optax (Babuschkin et al., 2020) with 10^{-3} learning rate, 50,000 training steps and 50 batch size as recommended by Wistuba and Grabocka (2021). In each iteration of Adam, we sample datapoints uniformly at random for each task, and compute the log likelihoods. Then, we sum up the log likelihoods across all tasks and use the sum as the loss function value.[6]

- H-EKL: HyperBO with a GP pre-trained via the EKL objective, i.e., Eq. 8 which sums over Eq. 7 (without the $\ln|\tilde{\Sigma}|$ term) on "matching-input" datasets. If the estimated covariance matrix is low rank, we report Eq. 6 as the EKL values for the "matching-input" datasets. We used the same GP model as H-NLL. We optimized EKL using L-BFGS (Liu and Nocedal, 1989) for 100 iterations.

- FSBO: "Few-Shot Bayesian optimization" proposed by Wistuba and Grabocka (2021). FSBO is a special case of HyperBO with zero mean and the NLL objective. We used the same kernel function, parameter warping and optimization method as H-NLL.

5. The reason of using $\max_{\tau \in [t-1]} y_\tau + 0.1$ was to approximate the max-value of the function as suggested by Wang and Jegelka (2017), and we found it to be effective across compared HyperBO variants. Because the observations are (log) error rates, this acquisition function trades off exploration and exploitation - i.e., with larger error rates this seeks relatively more substantial improvements than with small error rates. We also tested other acquisition functions and the results can be found in §D.2.
6. In practice, we found Adam performed better than L-BFGS on NLLs of GPs with Matérn52 kernel. This is partly because Matérn52 kernel with a lot of datapoints can be (numerically) low-rank given that the covariance matrix is represented with finite bits in a computer. We did not investigate further in this work and decided to use Adam for NLL. This problem did not seem to occur for EKL because of much fewer matching datapoints.

- ABLR: BO with "Adaptive Bayesian Linear Regression" proposed by Perrone et al. (2018). ABLR is a special case of HyperBO with zero mean and linear kernel $k(x, x') = b^2 + \phi(x)^T \phi(x')/s^2$ where s and b are kernel parameters, and ϕ is the feature layer of a 2-hidden-layer neural network of size $(32, 32)$. We used the same optimization method as H-NLL.

These settings of HyperBO use different mean and kernel structures in pre-training. We provide more comparisons over acquisition functions and other variants of the GP models in Appendix D.

For other meta BO baselines, we included two scalable methods, MIMO and RFGP, that replace the GP with a regression model that can be trained using stochastic gradient descent and thus scales linearly in the number of observations. Following the multi-task setup of Springenberg et al. (2016), we jointly trained a 5-dimensional embedding of each task, which was then added to the input of MIMO and RFGP. We also compared to MAF proposed by Volpp et al. (2020).

- MIMO: Multi-task BO with an ensemble of feedforward neural networks with shared sub-networks (Havasi et al., 2021; Kim et al., 2022) as the surrogate model. We used 1 shared dense layer of size 10 and 2 unshared layers of size 10. We used tanh activation based on Figure 2 from Snoek et al. (2015). The network has one output unit with linear activation and another with $\mathrm{softplus}(10^{-4}, 1)$ activation, corresponding respectively to the mean and standard deviation parameters of a normal distribution. In each BO iteration, we trained for 1000 epochs using the Adam optimizer with learning rate 10^{-4} and batch size 64.

- RFGP: Multi-task BO using a GP approximated by random features (Snoek et al., 2015; Krause and Ong, 2011). We used the open-source implementation of random Fourier features by Liu et al. (2020). In each BO iteration, we trained for 1000 epochs using the Adam optimizer with learning rate 10^{-3} and batch size 64.

- MAF: The "Meta Acquisition Function" method from Volpp et al. (2020). MAF used reinforcement learning to learn an acquisition function modeled by a neural network over a set of transfer learning tasks. All MAF results were generated using the code from Volpp et al. (2020). See App. D.5 for experimental details. As MAF takes significantly longer to run than HyperBO and other methods, we only include its results for §7.4.1 and Figure 13.

Unlike these meta BO baselines, model re-training (a.k.a. fine-tuning) is optional for HyperBO during BO iterations since pre-training is sufficient to obtain the GP prior. Unless otherwise mentioned, we did not use any model re-training for HyperBO variants, and posterior inference (for computing acquisition functions) was done without changing the GP in the BO module of HyperBO.

The remaining baselines are those that *do not* use information from training tasks:

- Rand: A random search method that samples uniformly randomly in the search space in each BO iteration. For hyperparameters with input warping, we sample from the corresponding warped input space. For example, as shown in Table 4, the learning rate η has a rectangular range $[10^{-5}, 10]$ and a logarithm warping function. So, instead of uniformly randomly sampling in $[10^{-5}, 10]$, Rand samples $\log_{10} \eta$ uniformly randomly in $[-5, 1]$.

- STBO: Single task BO with the same acquisition function and GP structure as H-NLL, except that we put a Lognormal$(0, 0.1)$ prior on the warped lengthscale and signal variance parameters of the anisotropic Matérn52 kernel, and a Normal$(0, 0.1)$ prior on the warped noise variance. In each BO iteration, STBO optimizes the GP hyperparameters by running L-BFGS for 100 iterations on the marginal likelihood of observations from the test task.

- STBOH: Single task GP-UCB (coefficient=1.8) with constant mean, anisotropic Matérn52 kernel and *hand-tuned* prior on GP hyperparameters and the UCB coefficient (Srinivas et al., 2010; Golovin et al., 2017). Specifically, log signal variance follows Normal(-1, 1), log lengthscale (one per input parameter) follows Normal$(0, 1)$, and log observation noise variance follows Normal(-6, 3). The GP hyperparameters are post-processed by tensorflow-probability's `SoftClip` bijector to constrain the values between 1st and 99th quantiles. These prior distributions were manually tuned to obtain reasonable convergence rates on 24 analytic functions in COCO (Hansen et al., 2021). The GP hyperparameters are optimized via maximum marginal likelihood in each BO iteration.

7.3 Reporting Aggregated Multi-task Results

For each test task, our goal is to obtain a low simple regret with as few BO iterations as possible. In many situations, we cannot compute the simple regret defined in §4 due to the lack of information on the test function. However, in the case of running BO on the offline data of PD1 and HPO-B, the test function has a finite set of datapoints and there exists $y_{\max}$, the maximum function value in this finite set. Thus, we can empirically report the regret at the t-th iteration of BO as $r_t = y_{\max} - \max_{\tau \in [t]} y_\tau$, where $[y_\tau]_{\tau=1}^t$ is the sequence of observed function values accumulated in BO.

In the experiments, we ran BO on a set of test tasks with multiple random seeds, and we obtained a collection of regrets over BO iterations: $\{\{[r_1^{(p,q)}, \cdots, r_T^{(p,q)}]\}_{p=1}^P\}_{q=1}^Q$, where P is the number of test tasks and Q is the number of random seeds. To report the performance of BO, we used the following three ways to aggregate the regrets and demonstrate how well each method performs.

Regret curves show how regrets change as the number of BO iterations increases. At the t-th iteration, we report the median and 20/80 percentile of $\{\frac{1}{P}\sum_{p=1}^P r_t^{(p,q)}\}_{q=1}^Q$, where each element $\frac{1}{P}\sum_{p=1}^P r_t^{(p,q)}$ is the mean of the regrets over test tasks.

Performance profiles (Dolan and Moré, 2002) have been widely used to evaluate the performance of optimization algorithms. We define the performance profile for a BO method as the fraction of tasks that the method solves at each iteration. The notion of "solving a task" depends on specific criteria. We set the criteria to be achieving a regret lower than a constant C at the t-th iteration of BO. More formally, the performance profile across BO iterations is $[\frac{1}{PQ}\sum_{q=1}^Q\sum_{p=1}^P \mathbb{1}_{r_1^{(p,q)}<C}, \cdots, \frac{1}{PQ}\sum_{q=1}^Q\sum_{p=1}^P \mathbb{1}_{r_T^{(p,q)}<C}]$.

Ranking plots show how the rank of a method (comparing to other competing methods) changes as the number of BO iterations increases. To obtain the ranks, we take the mean of regrets over tasks (in the same way as regret curves) and rank the compared methods from low to high. If there is a tie, we assign the average of the ranks to all the tied methods (e.g., if two methods obtain the same best regret value, their ranks are both 1.5). We report the mean and 1 standard deviation of the ranks over random seeds in ranking plots.

7.4 Results on the PD1 Offline Hyperparameter Tuning Tasks

Many tasks in §7.1 can use a lot of compute resources, which makes it infeasible to perform a wide variety of experiments to analyze the characteristics of BO methods. Hence we adopted an offline approximation, which sets the search space of a test task to be the finite set of points that the corresponding tuning sub-dataset contains. We also filtered the diverged training runs in PD1

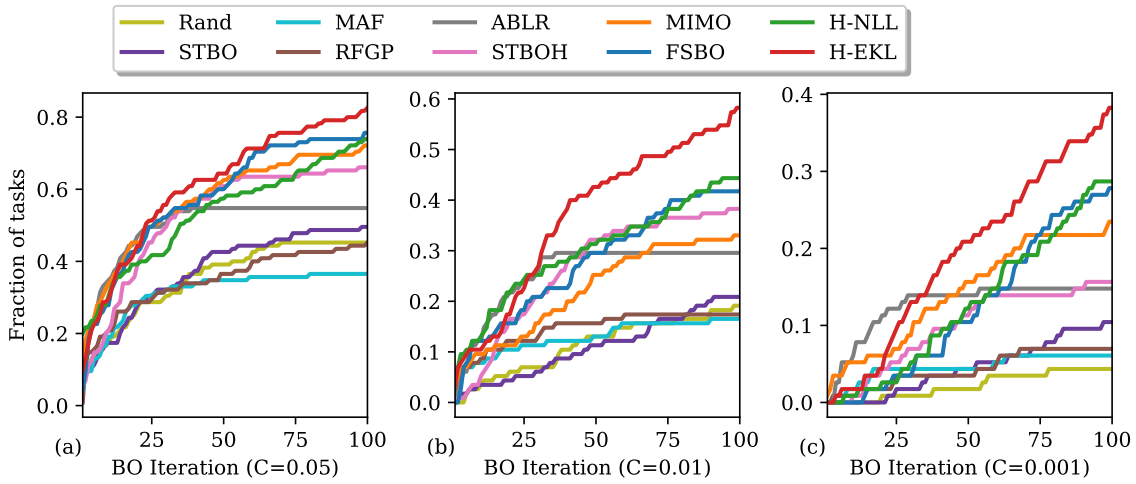

Figure 7: Performance profiles that show the fraction of tasks each method can solve, i.e., reach at most C amount of regret, in each BO iteration for the PD1 problem. In (a), C=0.05, in (b), C=0.01 and in (c), C=0.001. H-EKL was able to outperform other variants of HyperBO and other baselines.

to ensure the regrets can be computed. To simulate the online setting, the same datapoint can be observed multiple times, and we used zero initial data for all test tasks. Each method was repeated 5 times with different random seeds to initialize its model.

In the following, §7.4.1 demonstrates the overall performance of HyperBO; §7.4.2 presents the GP regression results and their impact on BO; §7.4.3 and §7.4.4 show the effects of the number of training datapoints and the number of training tasks.

7.4.1 HOLDING OUT RELEVANT TASKS

We first conducted experiments in a setting where a new task dataset is presented, and a BO method tunes the optimizer hyperparameters for a selected model on that task dataset with a specific hardware. A training dataset (see terminology in §4) is composed of tuning sub-datasets from at most 18 training tasks that do not involve the same task dataset as the test task. For training, H-EKL can only pre-train on the matched dataset while all other meta BO methods can access both the matched and unmatched datasets in PD1.

Figure 7 shows *performance profiles* of all compared methods described in §7.2. The performance profiles show the fraction of all test tasks that each method is able to solve by reaching at most 0.05, 0.01 and 0.001 regrets. The larger the fraction of tasks at each BO iteration, the better the method is. From all 3 criteria, we can see that HyperBO methods, especially H-EKL solved more tasks than other methods. As the performance criteria becomes more stringent in Figure 7(b) and Figure 7(c), the HyperBO variants, H-NLL and FSBO, also outperformed the baselines.

Figure 8 illustrates the regret curves, together with the vertical slices at the 1st and 100th iterations. Rand fell behind most BO alternatives but outperformed MAF, RFGP and STBO in terms of average regrets. However, it is worth noticing that Figure 7(c) shows MAF, RFGP and STBO can solve more tasks than Rand. ABLR also obtained high average regrets, but it solved more tasks than Rand for all criteria in Figure 7. Among all transfer learning BO methods, MAF and RFGP did not seem to

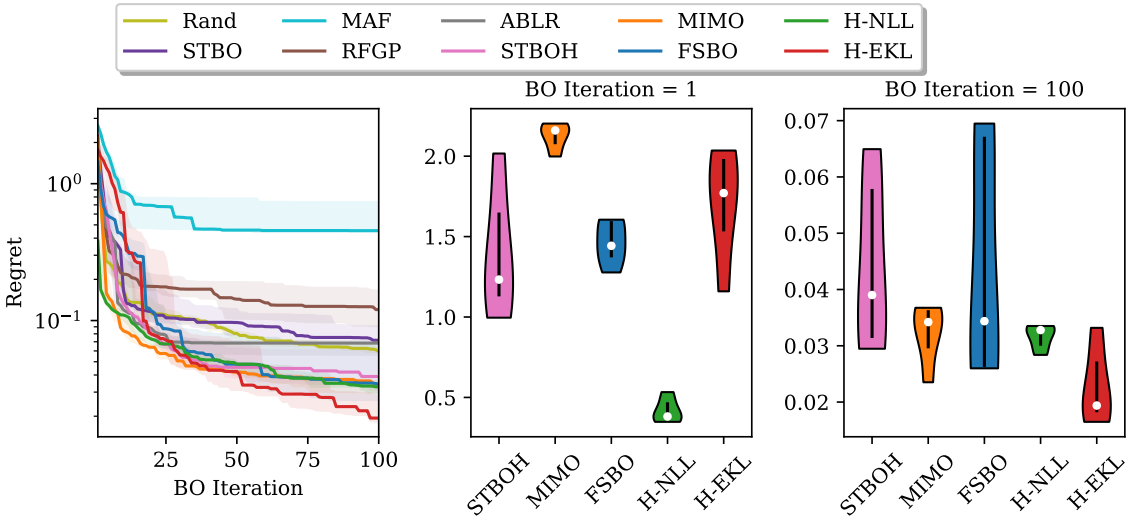

Figure 8: Regret curves for the PD1 problem. The violin plots (the middle and right plots) show the two vertical slices of the regret curves at the 1st and 100th iterations, where each white dot is the median and each black line is the 20/80 percentile. The HyperBO variants, H-NLL, H-EKL and FSBO, performed competitively comparing to other methods. In particular, H-NLL achieved much lower regrets than other methods in the beginning BO iterations.

benefit from multi-task data, given their similar or worse performance than the single task method STBO. One problem with STBO is that it trains the GP on the data that the GP suggests to query, which is easy to overfit (as shown by Table 2 in §6.3.2) if the priors on GP hyperparameters are not set carefully. With carefully hand-tuned priors, STBOH obtained better performance than STBO, ABLR, RFGP and MAF, showing the benefits of good priors designed with strong expert knowledge. MIMO surpassed STBOH, showing the potential of transfer learning in absence of expert knowledge. H-EKL did not locate good points in the beginning iterations, but found better points than other methods in later iterations; this was likely because our search spaces for mean and kernel functions encouraged the EKL objective (Eq. 5) to overfit the mean function while matching the covariance matrix, which helps the BO performance once enough datapoints are collected.

Figure 9 presents the ranking plot. Among all HyperBO variants, H-EKL was able to outperform all competing methods with BO iterations larger than 50. ABLR used a much more constrained kernel function than other HyperBO variants, resulting in worse performance. MIMO started off with a lower ranking, but it adapted well by re-training on new observations. The performance difference between H-NLL and FSBO shows that using a relatively complex mean function (as in H-NLL) can bring gains in performance comparing to using a zero mean (as in FSBO), especially in early stages of BO.

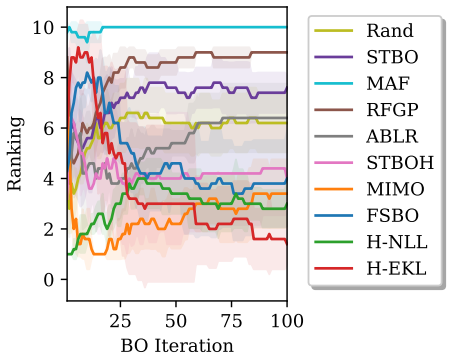

Figure 9: The ranking plot for PD1.

Speedup. To quantify HyperBO's advantage in terms of sample efficiency, we computed how much faster HyperBO can get a lower regret than the best alternative method; i.e., the median BO iterations needed for the best alternative to reach its lowest regret divided by the median BO iterations needed for HyperBO to surpass that best alternative. We found that on average, H-NLL and H-EKL were at least 3 times faster than the best non-HyperBO method and at least 7 times faster than Rand for the majority of the test tasks.

7.4.2 CORRELATIONS BETWEEN THE TEST NLL, PRE-TRAINING LOSS AND BO PERFORMANCE

Continuing the setup of §7.4.1 that holds out relevant tasks during pre-training, we explore how the pre-training loss correlates with the NLL on test data and BO performance. The pre-training losses, including EKL and NLL, are also measures of the GP regression performance. As a reference, we include quantitative results in Table 5 and Table 6. Table 5 shows the task-wise best validation error rates obtained by the top 6 methods in 100 BO iterations, and Table 6 shows the Test NLL (Eq. 14 on all available data of the test task in addition to observations) and EKL (Eq. 8 computed by summing Eq. 6 on the matched data constructed from both training and test data) for each test task. Table 5 again confirms the importance of using more expressive mean and kernel functions for pre-trained GPs. In particular, H-EKL achieved the lowest error rates on over half of the test tasks.

Figure 10 shows the positive correlations between the simple regret, Test NLL, NLL (Eq. 13 on irrelevant tasks) and EKL. Each point in the plots of Figure 10 corresponds to an experiment, where for each test task, we pre-train a GP with random initialization, and we evaluate the metrics averaged over all test tasks. We repeated such experiment 5 times and hence there are 5 sets of regret and loss values for each method in Figure 10. Together with Table 5 and Table 6, we can tell that optimizing the NLL or EKL losses leads to a lower Test NLL, which correlates with a lower regret. Moreover, as shown in Table 6, H-EKL may reach a lower Test NLL faster than H-NLL. And H-NLL may reach a lower EKL, which is evaluated on both training and test tasks, than H-EKL.

7.4.3 EFFECTS OF THE NUMBER OF TRAINING DATAPOINTS

In this section, we answer the question: how does M_i in §4, the number of datapoints in each training task, impact the performance of meta BO methods? We used data from all but the test task for meta BO methods, with the maximum number of training datapoints per task ranging from 4 to about 1600. The training datapoints were selected uniformly randomly from tuning sub-datasets. Since there are in total 242 valid "matching-input" datapoints across training tasks in PD1, we only show the regrets of H-EKL up to 242 datapoints per training task. We did not include MAF in the following experiments due to its high computational costs.

Figure 11 shows how the simple regrets (aggregated over 23 test tasks at the 100th BO iteration) change as we increase the maximum number of datapoints per training task ($\max_{i\in[N]} M_i$). Not surprisingly, with only 4 datapoints per training task, all meta BO methods obtained worse performance than STBOH. As the number of datapoints increased to about 100, H-EKL outperformed STBOH by a large margin. And with about 200 datapoints per training task or more, both H-EKL and H-NLL surpassed STBOH. We observed clear trend that more training datapoints can lead to lower regrets for H-EKL, H-NLL and ABLR. FSBO and MIMO benefited from more training data but showed more variations in regrets after $\max_{i\in[N]} M_i$ reached about 100.

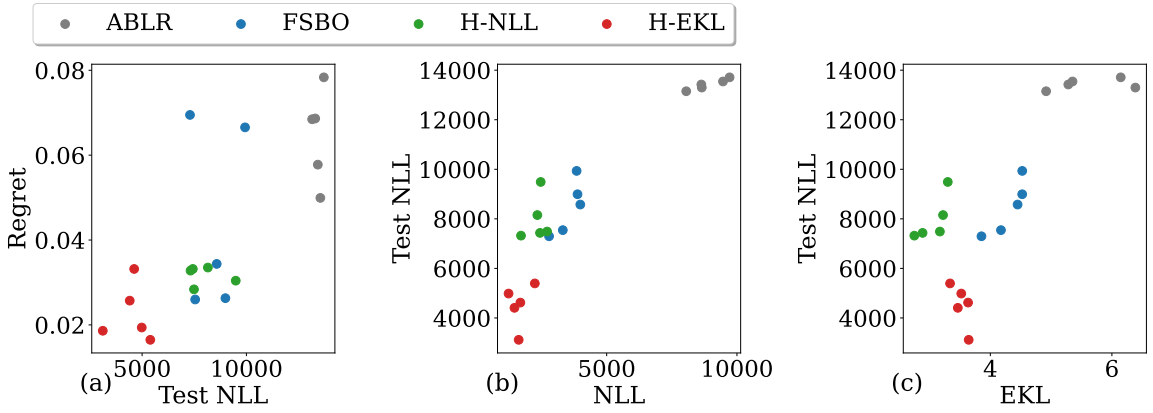

Figure 10: Correlations between (a) the simple regret at the 100th BO iteration and Test NLL, whose Pearson correlation coefficient is 0.77; (b) Test NLL and NLL for the tasks irrelevant to the test task, whose Pearson correlation coefficient is 0.93; and (c) Test NLL and EKL on all "matching-input" datapoints, whose Pearson correlation coefficient is 0.80. Each point corresponds to a random repeat of the experiment, with metrics averaged over the test tasks. These plots confirm positive correlations between the simple regret, Test NLL, NLL and EKL, showing more evidence that optimizing the NLL (Eq. 13) and EKL (Eq. 8) loss functions can bring better BO performance.

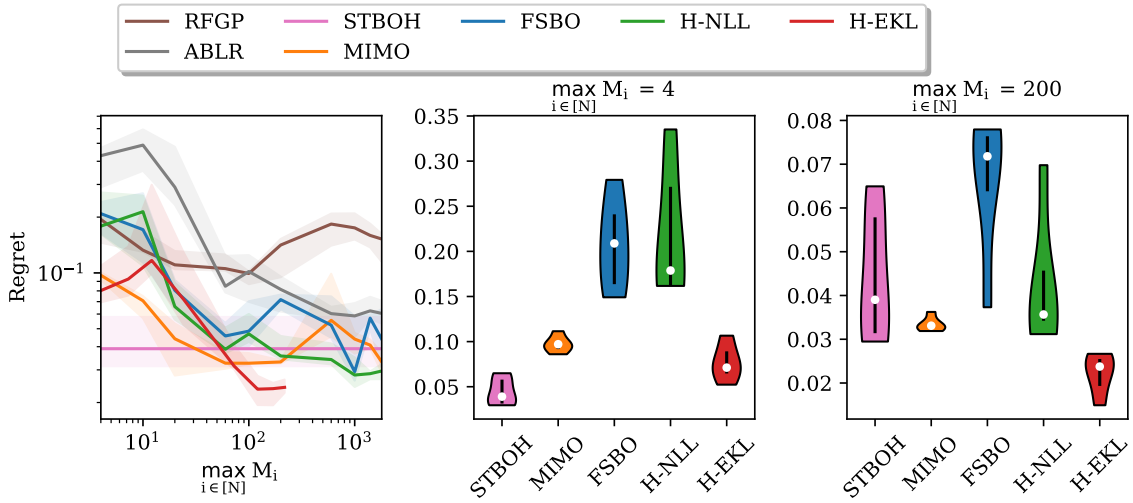

Figure 11: The median and 20/80 percentiles of simple regrets as the maximum number of datapoints per training task ($\max_{i\in[N]} M_i$) changes from 4 to about 1600 for each method. We also show the two vertical slices for $\max_{i\in[N]} M_i = 4$ (middle) and $\max_{i\in[N]} M_i = 200$ (right). H-EKL used significantly fewer training datapoints to obtain lower regrets than other meta BO methods.

	ABLR	STBOH	MIMO	FSBO	H-NLL	H-EKL
WMT XFormer 64	**33.91 ± 0.02**	34.17 ± 0.37	34.27 ± 0.34	33.94 ± 0.03	**33.91 ± 0.02**	33.95 ± 0.02
Uniref50 Transformer 128	79.19 ± 0.69	78.92 ± 0.27	79.00 ± 0.72	78.81 ± 0.22	78.85 ± 0.26	**78.72 ± 0.17**
LM1B Transformer 2048	61.85 ± 0.05	61.95 ± 0.08	61.96 ± 0.11	**61.81 ± 0.02**	**61.81 ± 0.03**	61.83 ± 0.03
SVHN WRN 1024	4.07 ± 0.01	4.12 ± 0.17	**3.89 ± 0.09**	4.07 ± 0.00	4.07 ± 0.00	4.07 ± 0.00
SVHN WRN 256	3.73 ± 0.03	3.72 ± 0.06	**3.61 ± 0.06**	3.74 ± 0.04	3.76 ± 0.00	3.69 ± 0.02
ImageNet ResNet50 256	22.66 ± 0.06	22.64 ± 0.16	22.82 ± 0.20	22.63 ± 0.00	22.63 ± 0.00	**22.53 ± 0.00**
ImageNet ResNet50 512	22.73 ± 0.05	22.71 ± 0.11	22.92 ± 0.12	**22.63 ± 0.00**	22.71 ± 0.06	22.78 ± 0.03
MNIST CNNPoolTanh 2048	0.55 ± 0.06	0.53 ± 0.02	0.52 ± 0.02	**0.51 ± 0.00**	**0.51 ± 0.00**	0.53 ± 0.01
MNIST CNNPoolTanh 256	0.47 ± 0.01	0.47 ± 0.02	0.47 ± 0.01	**0.45 ± 0.00**	**0.45 ± 0.00**	0.47 ± 0.01
MNIST CNNPoolReLU 2048	0.74 ± 0.02	0.74 ± 0.04	**0.67 ± 0.02**	0.69 ± 0.05	0.71 ± 0.03	0.69 ± 0.03
MNIST CNNPoolReLU 256	0.51 ± 0.01	0.55 ± 0.06	0.51 ± 0.02	0.50 ± 0.01	0.50 ± 0.00	**0.48 ± 0.01**
MNIST CNNReLU 2048	1.36 ± 0.55	1.19 ± 0.18	1.09 ± 0.02	1.23 ± 0.18	1.14 ± 0.08	**1.06 ± 0.01**
MNIST CNNReLU 256	1.07 ± 0.06	1.05 ± 0.03	1.07 ± 0.03	1.40 ± 0.41	**1.03 ± 0.00**	**1.03 ± 0.00**
Fashion CNNPoolTanh 2048	7.20 ± 0.07	7.06 ± 0.14	7.10 ± 0.02	7.04 ± 0.07	7.04 ± 0.07	**7.03 ± 0.11**
Fashion CNNPoolTanh 256	6.52 ± 0.05	6.72 ± 0.38	6.42 ± 0.17	6.38 ± 0.10	6.28 ± 0.03	**6.26 ± 0.00**
Fashion CNNPoolReLU 2048	7.62 ± 0.05	**7.45 ± 0.10**	7.55 ± 0.02	7.48 ± 0.03	7.50 ± 0.05	7.50 ± 0.06
Fashion CNNPoolReLU 256	6.77 ± 0.09	6.75 ± 0.04	7.02 ± 0.04	6.76 ± 0.07	6.79 ± 0.11	**6.70 ± 0.05**
Fashion CNNReLU 2048	8.52 ± 0.94	7.56 ± 0.02	7.60 ± 0.07	7.73 ± 0.36	7.63 ± 0.12	**7.45 ± 0.19**
Fashion CNNReLU 256	8.37 ± 0.71	7.41 ± 0.23	7.54 ± 0.23	7.58 ± 0.51	**7.36 ± 0.40**	7.48 ± 0.54
CIFAR100 WRN 2048	21.79 ± 0.32	20.78 ± 0.43	20.90 ± 0.65	20.87 ± 0.11	21.22 ± 0.37	**20.48 ± 0.07**
CIFAR100 WRN 256	19.20 ± 0.23	19.03 ± 0.09	19.36 ± 0.20	**18.99 ± 0.02**	19.00 ± 0.00	19.03 ± 0.07
CIFAR10 WRN 2048	3.81 ± 0.21	3.42 ± 0.16	3.54 ± 0.16	3.38 ± 0.04	3.49 ± 0.12	**3.35 ± 0.05**
CIFAR10 WRN 256	2.85 ± 0.06	2.88 ± 0.14	2.77 ± 0.10	2.83 ± 0.03	2.85 ± 0.01	**2.76 ± 0.05**

Table 5: The mean and standard deviation of best validation error rates (%) for each test task in the PD1 problem. Meta BO methods including MIMO and HyperBO variants (H-NLL, H-EKL, FSBO, ABLR) have access to training tasks that do not share the same task dataset as the test task. We show results of the top 6 methods, and we highlight the lowest error rates in bold.

	Test NLL				EKL			
	ABLR	FSBO	H-NLL	H-EKL	ABLR	FSBO	H-NLL	H-EKL
WMT XFormer 64	−25 ± 2	**−32 ± 1**	−32 ± 2	−31 ± 1	7.6 ± 2.7	5.1 ± 1.3	3.5 ± 0.8	**3.5 ± 0.2**
Uniref50 Transformer 128	−30 ± 2	**−36 ± 2**	−36 ± 2	−35 ± 1	7.6 ± 3.1	4.9 ± 1.0	3.7 ± 1.3	**3.4 ± 0.2**
LM1B Transformer 2048	−27 ± 1	**−32 ± 2**	−31 ± 2	−30 ± 1	6.4 ± 1.8	5.0 ± 1.2	**3.4 ± 0.9**	3.5 ± 0.1
SVHN WRN 1024	301 ± 71	333 ± 70	335 ± 84	**44 ± 10**	6.7 ± 2.7	4.5 ± 0.8	**3.3 ± 0.6**	3.5 ± 0.2
SVHN WRN 256	294 ± 55	308 ± 82	311 ± 108	**12 ± 11**	6.7 ± 2.7	4.5 ± 0.8	**3.3 ± 0.6**	3.5 ± 0.1
ImageNet ResNet50 256	1 ± 3	**−23 ± 1**	−21 ± 1	−16 ± 2	4.4 ± 0.2	3.9 ± 0.1	**2.7 ± 0.1**	3.6 ± 0.1
ImageNet ResNet50 512	−2 ± 6	**−22 ± 1**	−20 ± 1	−15 ± 2	4.4 ± 0.1	3.9 ± 0.1	**2.7 ± 0.1**	3.6 ± 0.1
MNIST CNNPoolTanh 2048	165 ± 17	83 ± 13	68 ± 5	**34 ± 4**	5.0 ± 0.3	4.1 ± 0.2	**2.7 ± 0.1**	3.4 ± 0.1
MNIST CNNPoolTanh 256	199 ± 22	51 ± 13	45 ± 11	**38 ± 4**	4.7 ± 0.3	4.0 ± 0.1	**2.7 ± 0.1**	3.4 ± 0.2
MNIST CNNPoolReLU 2048	592 ± 30	537 ± 59	483 ± 13	**425 ± 111**	9.4 ± 1.3	6.3 ± 0.6	5.4 ± 0.6	**3.5 ± 0.2**
MNIST CNNPoolReLU 256	278 ± 25	66 ± 33	45 ± 5	**28 ± 10**	5.0 ± 0.7	4.0 ± 0.3	**2.8 ± 0.1**	3.5 ± 0.1
MNIST CNNReLU 2048	386 ± 22	280 ± 37	**276 ± 38**	289 ± 67	6.9 ± 1.2	4.6 ± 0.4	**3.5 ± 0.4**	3.6 ± 0.1
MNIST CNNReLU 256	275 ± 21	155 ± 12	**138 ± 9**	245 ± 43	4.9 ± 0.1	4.4 ± 0.3	**3.0 ± 0.1**	3.8 ± 0.1
Fashion CNNPoolTanh 2048	60 ± 6	0 ± 1	3 ± 5	**−6 ± 2**	5.1 ± 0.4	3.9 ± 0.2	**2.6 ± 0.0**	3.5 ± 0.2
Fashion CNNPoolTanh 256	45 ± 7	11 ± 2	13 ± 3	**1 ± 1**	5.0 ± 0.2	3.9 ± 0.2	**2.6 ± 0.0**	3.5 ± 0.2
Fashion CNNPoolReLU 2048	82 ± 4	26 ± 4	24 ± 4	**9 ± 2**	5.0 ± 0.2	3.9 ± 0.2	**2.6 ± 0.0**	3.5 ± 0.2
Fashion CNNPoolReLU 256	56 ± 8	9 ± 4	9 ± 4	**−8 ± 1**	4.9 ± 0.2	3.9 ± 0.2	**2.6 ± 0.0**	3.5 ± 0.2
Fashion CNNReLU 2048	75 ± 3	37 ± 6	**29 ± 3**	43 ± 6	5.1 ± 0.4	3.9 ± 0.2	**2.6 ± 0.0**	3.5 ± 0.2
Fashion CNNReLU 256	53 ± 9	15 ± 4	**12 ± 2**	29 ± 9	5.1 ± 0.3	3.9 ± 0.2	**2.6 ± 0.0**	3.5 ± 0.2
CIFAR100 WRN 2048	10 ± 5	−7 ± 3	−6 ± 2	**−10 ± 1**	5.2 ± 1.0	4.3 ± 0.9	**3.2 ± 0.8**	3.6 ± 0.1
CIFAR100 WRN 256	0 ± 3	−10 ± 2	−10 ± 2	**−12 ± 2**	5.3 ± 0.9	4.3 ± 0.9	**3.2 ± 0.8**	3.6 ± 0.1
CIFAR10 WRN 2048	187 ± 61	137 ± 50	148 ± 69	**1 ± 4**	4.4 ± 0.2	4.0 ± 0.2	**2.7 ± 0.2**	3.6 ± 0.1
CIFAR10 WRN 256	114 ± 31	63 ± 18	53 ± 13	**1 ± 3**	4.4 ± 0.2	4.0 ± 0.2	**2.7 ± 0.2**	3.6 ± 0.1

Table 6: The mean and standard deviation of Test NLL ($\times 100$) and EKL for each test task in the offline optimizer hyperparameter tuning experiments. HyperBO variants only have access to training tasks that do not share the same task dataset as the test task. We highlight the lowest Test NLL and the lowest EKL in bold.

7.4.4 EFFECTS OF THE NUMBER OF TRAINING TASKS

We investigate the impact of the number of training tasks on the performance of meta BO methods in this section. Figure 12 demonstrates how the regrets change as we increase the number of training tasks. We show the aggregated simple regrets of meta BO methods at the 100th BO iteration over the 23 test tasks. To reduce training tasks, we first remove the tasks that involve the same task dataset

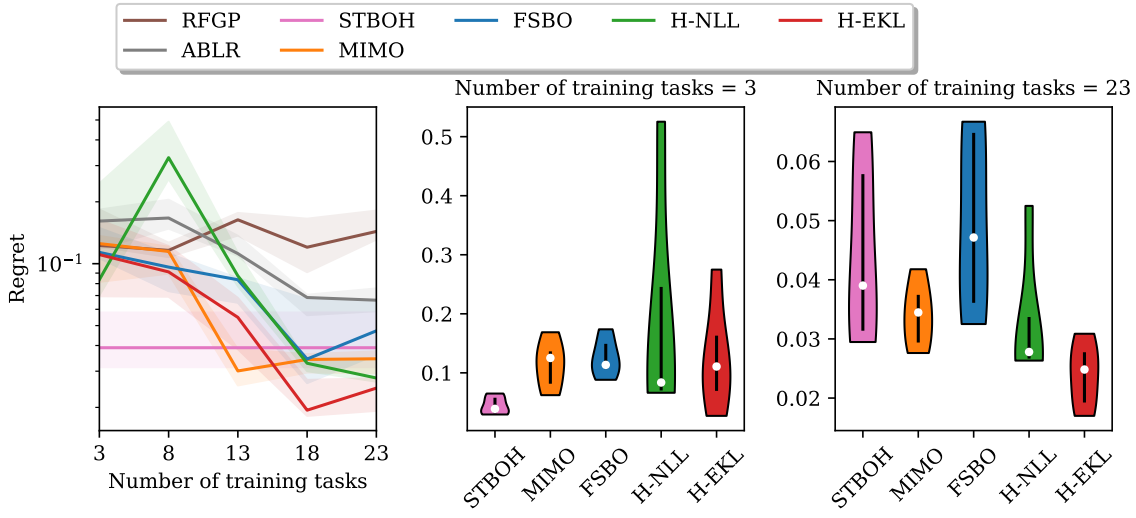

Figure 12: The median and 20/80 percentiles of simple regrets for each method with 3 to 23 training tasks. The violin plots for the top performing methods are shown for 3 and 23 training tasks, where each white dot is the median and black line the 20/80 percentiles. All meta BO methods except RFGP got lower regrets with more training tasks. H-NLL and H-EKL achieved better performance than other baselines with 23 training tasks.

as the test task, and then remove others randomly until we reach the designated number of training tasks. This is to analyze the performance of all methods on less-related tasks. The models were never trained on the data from the test task.

All meta BO methods except RFGP reduced simple regrets as more training tasks were given. In both Figure 11 and Figure 12 , RFGP curiously did not seem to be able to learn from multi-task data. The worse performance of RFGP was likely caused by the high confidence predictions from random feature based Bayesian linear regression models as discussed by Wang et al. (2018a); overly confident predictions can lead to less exploration in BO and as a result, poor BO performance. ABLR also uses Bayesian linear regression and likely suffered from the same lack of exploration issue. With 18 or more training tasks, H-EKL and H-NLL outperformed the carefully hand-tuned STBOH. Figure 13 shows the superior performance of H-EKL and H-NLL comparing to other baselines in the setting where 23 training tasks are given.

7.5 Results on the PD1 Online Hyperparameter Tuning Tasks

In this section, we look into the online hyperparameter tuning setting where we deploy HyperBO as a BO service and optimize over the full hyperrectangular search space detailed in Table 4. In the online setting, some combinations of hyperparameters may be *infeasible* to evaluate. For example, an overly big learning rate may lead to divergence in gradients, in which case we do not obtain a valid model. To address this, we pre-process the function values to $[-2, 2)$ such that *infeasible* evaluations map to -2, while bad evaluations approach asymptotically to -2. More precisely, for each training

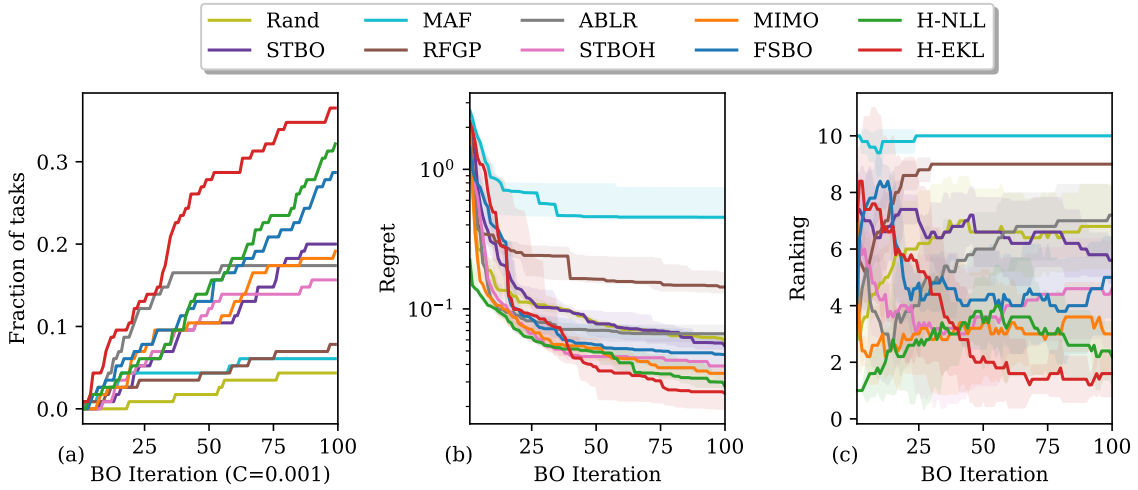

Figure 13: (a-c) are the performance profiles, regret curves and ranking plots for meta BO methods with 23 training tasks. As shown in (c), H-NLL is the best performing method in the first 30 BO iterations and H-EKL becomes the best afterwards. Both H-EKL and H-NLL outperformed baselines.

sub-dataset D_{f_i}, we apply for each successful $y \in \{y_j^{(i)}\}_{j=1}^{M_i}$ the following mapping:

$$y \leftarrow \frac{\text{softplus}(y - \overline{y})}{\text{softplus}(y_{\max} - \overline{y})} * 4 - 2$$

where $\overline{y}$ is the median and $y_{\max}$ is the maximum of $\{y_j^{(i)}\}_{j=1}^{M_i}$. In every iteration t of HyperBO in Algorithm 1, we also apply the same type of mapping to the test sub-dataset $D_f = \{(x_\tau, y_\tau)\}_{\tau=1}^{t-1}$. For all methods that we compared, model re-training was performed on D_f to allow this dynamic adjustment of the evaluations on the test function.

Figure 14 presents the online tuning results for image and text based tasks[7]. The goals of image based tasks are to minimize the validation error rate, while the text based tasks (LM1B, Uniref50, WMT) aim to minimize the cross entropy (CE) loss.

We reconfigured H-NLL and H-EKL to use a one-hidden layer neural network of size 8 as mean function and Matérn32 covariance function on the feature layer of the neural net as kernel. For H-EKL, we added 0.1 times the NLL objective to the EKL objective to allow re-training the model during online evaluations. For all methods, we used GP-UCB with coefficient 1.8.

In our experiments, we noticed that it was very difficult for STBO, MIMO and RFGP to recover from a "bad" datapoint, and their BO results were a lot worse than STBOH, H-NLL and H-EKL. This was partly because predictions from these models were significantly tied to the initial observations. For example, STBO may overfit to the initial bad value and believe there are bad values in the entire search space. Figure 14 presents the results of STBOH, H-NLL and H-EKL for clarity and Figure 22 shows the results that compare more methods.

Figure 14 shows the robust performance of H-NLL and H-EKL in the online tuning problems. In 8 out of 9 tuning tasks, HyperBO methods outperformed STBOH.

7. We only experimented on the 9 tasks selected from Table 3 due to limited compute resources.

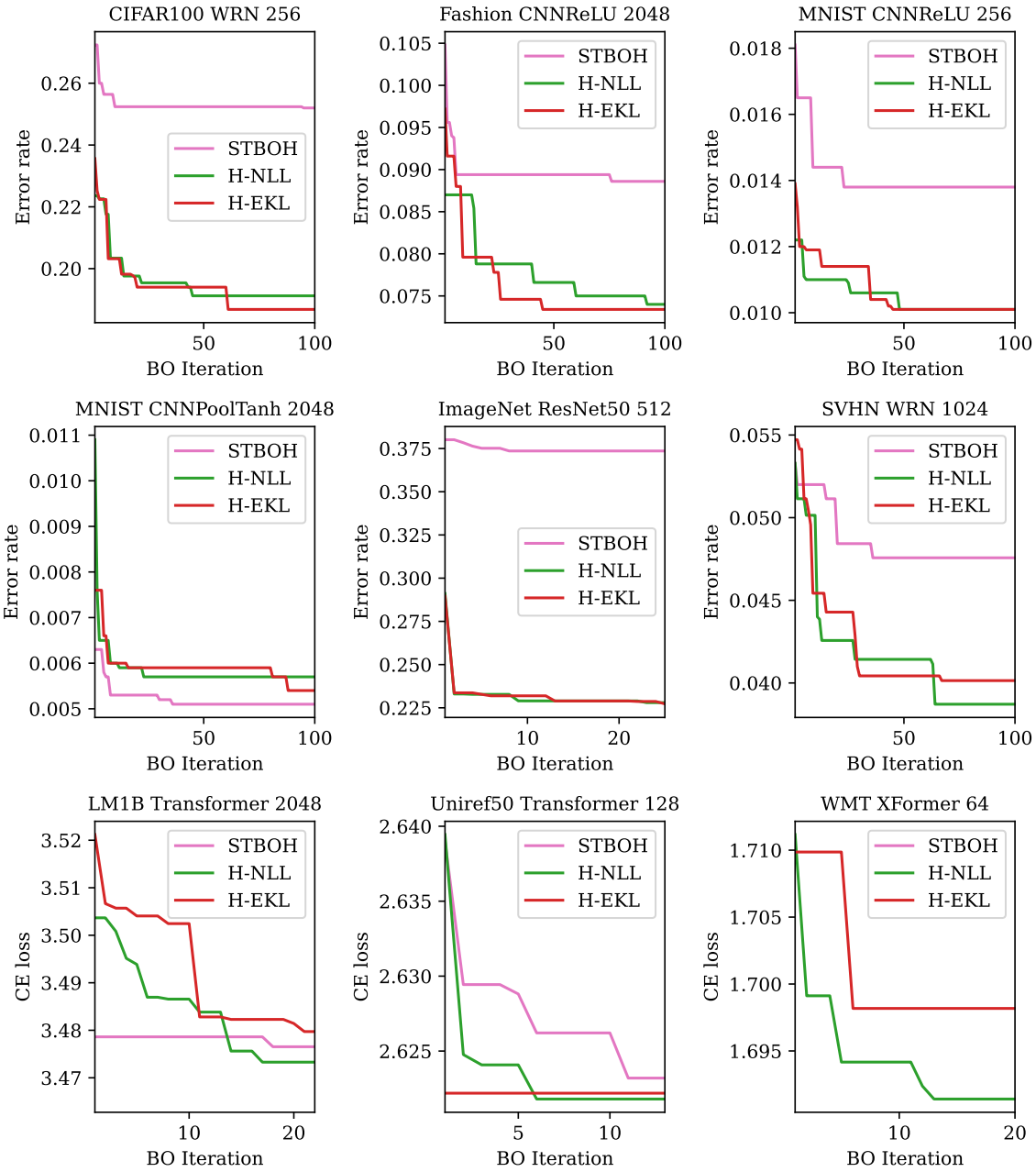

Figure 14: PD1 online hyperparameter tuning results for image based tasks with the best validation error rate as the objective, and text based tasks that use the best validation cross entropy (CE) loss as the objective. HyperBO methods achieved better performance in 8 out of 9 online tuning tasks. The CE losses of STBOH are not shown for WMT XFormer 64 because all of the datapoints acquired by STBOH on this task were infeasible.

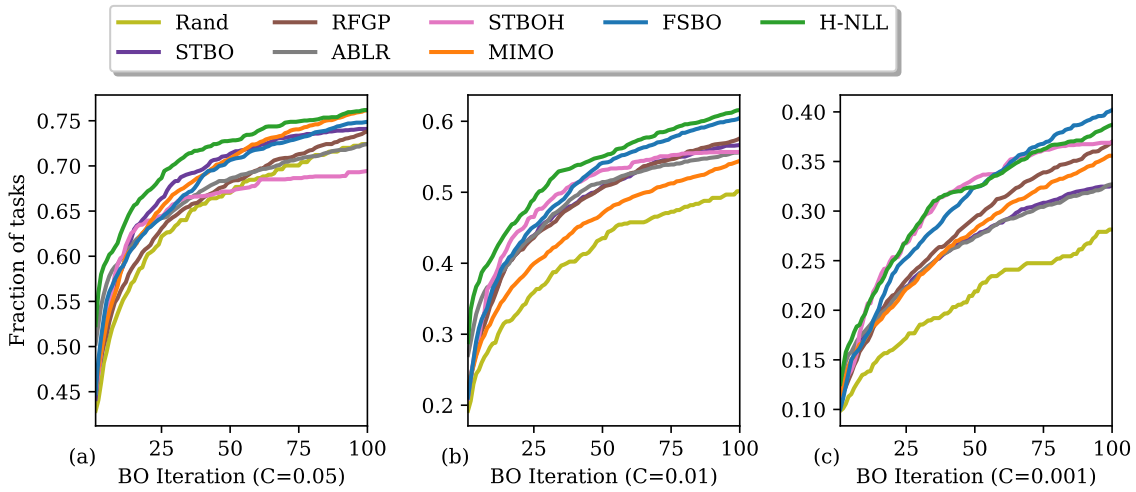

Figure 15: Performance profiles that show the fraction of tasks each method can solve, i.e., reach at most C amount of regret, in each BO iteration for the HPO-B experiment. In (a), C=0.05, in (b), C=0.01 and in (c), C=0.001. H-NLL and FSBO performed better than or similar to the best non-HyperBO alternatives. H-NLL solved more tasks than other methods for C=0.05 and C=0.01.

7.6 Results on the HPO-B Benchmark

In this experiment, we evaluate the performance of HyperBO on the HPO-B benchmark (Pineda-Arango et al., 2021). The HPO-B benchmark is a machine learning hyperparameter tuning dataset, which includes about 6 million evaluations of hyperparameters from 16 search spaces of different models. Each search space has different sets of hyperparameters with dimensions ranging from 2 to 18. There are multiple tasks in each search spaces, which are divided to training and test tasks. In total, there are 86 test tasks. Please refer to Pineda-Arango et al. (2021) for more details.

Following Wistuba and Grabocka (2021), we used 2-hidden layer neural nets of size (128, 128) to replace the neural nets of HyperBO variants, and normalized all the regrets by the range of function values of each test task. The BO experiments were conducted with the original function values. The experiments were repeated 25 times with 5 different random seeds, and for each seed, 5 different test sub-dataset initialization settings specified in the HPO-B dataset. Similar to §7.4, we adopted the offline evaluation setup on finite sets of datapoints in the test tasks of HPO-B. Due to the lack of matched datasets, we excluded H-EKL.

In the following, §7.6.1 shows aggregated BO results on the original HPO-B. §7.6.2 aims to understand the impacts of reduced training data on HyperBO and other transfer learning BO methods. §7.6.3 presents both the regret and GP regression results to investigate the "negative transfer" (Rothfuss et al., 2021) effect.

7.6.1 PRE-TRAINING ON THE FULL HPO-B DATASET

We first used the full HPO-B dataset from each search space for meta BO methods. Figure 15 shows the performance profiles. The two HyperBO variants, H-NLL and FSBO achieved top performance. H-NLL was able to solve more tasks than other methods for the success criteria C=0.05 and C=0.01. For C = 0.001, FSBO solved slightly more tasks than H-NLL after about 60 BO iterations.

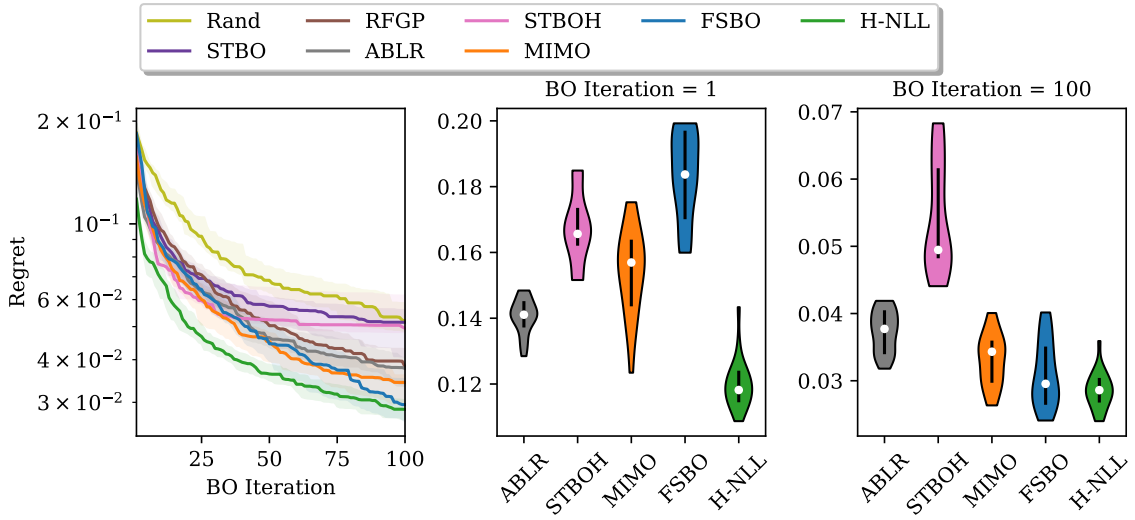

Figure 16: Regret curves for the HPO-B experiment. The violin plots (the middle and right plots) show the two vertical slices of the regret curves at the 1st and 100th iterations, where each white dot is the median and each black line is the 20/80 percentile. H-NLL achieved the lowest regrets across all BO iterations, outperforming FSBO, a HyperBO variant with 0 mean. For this experiment, having a more flexible mean function, as in H-NLL, helped in obtaining better performance.

Figure 16 presents the regret curves of all methods. All BO methods outperformed Rand, and all meta BO methods performed better than single task BO methods (STBO and STBOH) after about 50 BO iterations. H-NLL was able to locate better hyperparameters than all other methods across BO iterations, demonstrating the sample efficiency of HyperBO.

Figure 17 shows the corresponding ranking plot. H-NLL consistently ranked 1st across all iterations. H-NLL and FSBO are both HyperBO variants, and the more flexible mean function enabled H-NLL to acquire datapoints with lower regrets sooner than FSBO.

STBOH is equipped with a carefully hand-tuned prior, and it obtained lower regrets than other methods (except H-NLL) in some early iterations. Unlike the results in §7.4, the performance of STBOH plateaued quickly in the HPO-B experiment. It can be difficult for hand-tuned priors to generalize across a variety of tasks, and it is impractical to require more hand-tuning for new tasks. HyperBO provides an automatic model pre-training procedure to efficiently obtain customized priors, making this prior specification process much easier and more effective.

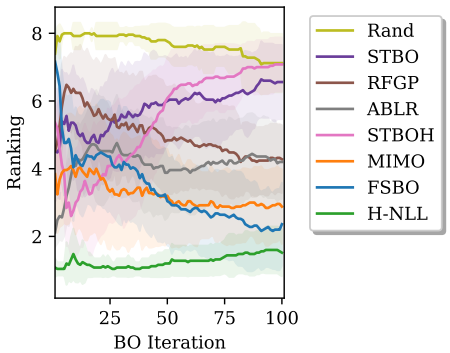

Figure 17: Ranking the compared methods for the HPO-B experiment.

Speedup. To compactly quantify the performance, we computed the speedup of HyperBO methods in the same way as was done in §7.4.1. We found

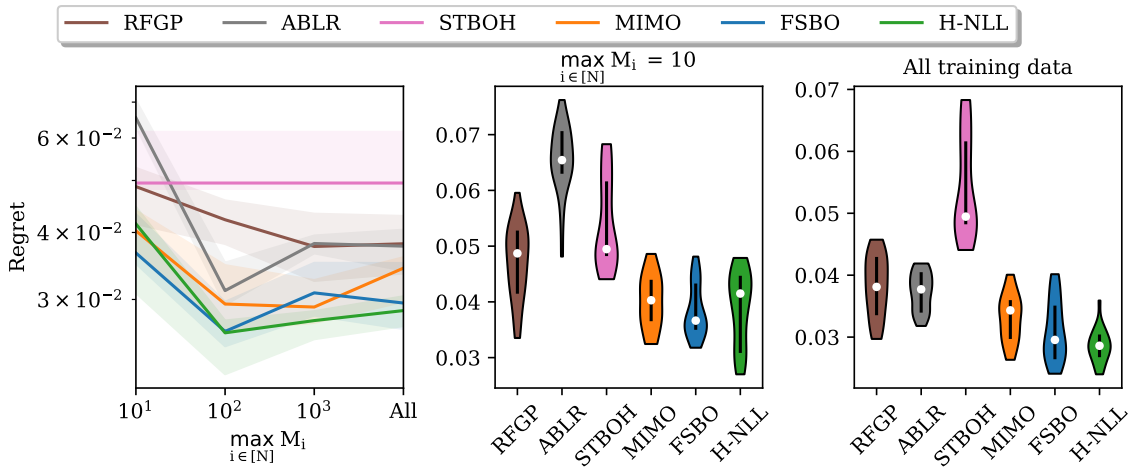

Figure 18: The median and 20/80 percentiles of simple regrets for meta BO methods as the maximum number of datapoints per training task in HPO-B ($\max_{i \in [N]} M_i$) increases. We randomly down-sampled each training sub-dataset to 10, 100, 1000 datapoints or used all training data for transfer learning. H-NLL and ABLR had the most amount of improvements when increasing the number of training datapoints per task from 10 to 100.

that on average, H-NLL was at least 6 times faster than FSBO, 10 times faster than the best non-HyperBO alternative, and 16 times faster than Rand for the majority of the test tasks.

7.6.2 PRE-TRAINING ON HPO-B WITH REDUCED TRAINING DATA

In this section, we investigate how the performance of pre-trained GPs changes as the number of training datapoints decreases for the tasks in HPO-B. To answer this question, we downsampled the HPO-B training datasets in each search space. For each training sub-dataset, we uniformly randomly sampled 10, 100 and 1000 training datapoints[8] for the meta BO methods to perform transfer learning.

Figure 18 shows the simple regrets at the 100th BO iteration of the meta BO methods that use different numbers of training datapoints per task, together with STBOH as a reference. All meta BO methods benefited from more training data as we increased the number from 10 per task to 100 per task. H-NLL and FSBO both obtained relatively better performance overall, followed by MIMO. RFGP and ABLR performed worse than other meta BO methods, similar to what we observed from a similar experiment for PD1 in Figure 11.

For the cases of above 100 datapoints per task, the performance of HyperBO variants and MIMO slightly worsened. Figure 19 compares the regret curves of meta BO methods with max 100 datapoints per training task versus with all training data in HPO-B. Meta BO methods with max 100 training datapoints per task interestingly unanimously outperformed their counterparts with the full training dataset, indicating redundancy and noise in the HPO-B dataset.

8. Among the 16 search spaces in HPO-B, 5 search spaces have at most 1000 datapoints (ranging from 259 to 895) in their training tasks. In other search spaces, the maximum number of datapoints for their corresponding training tasks ranges from 1,593 to 102,306.

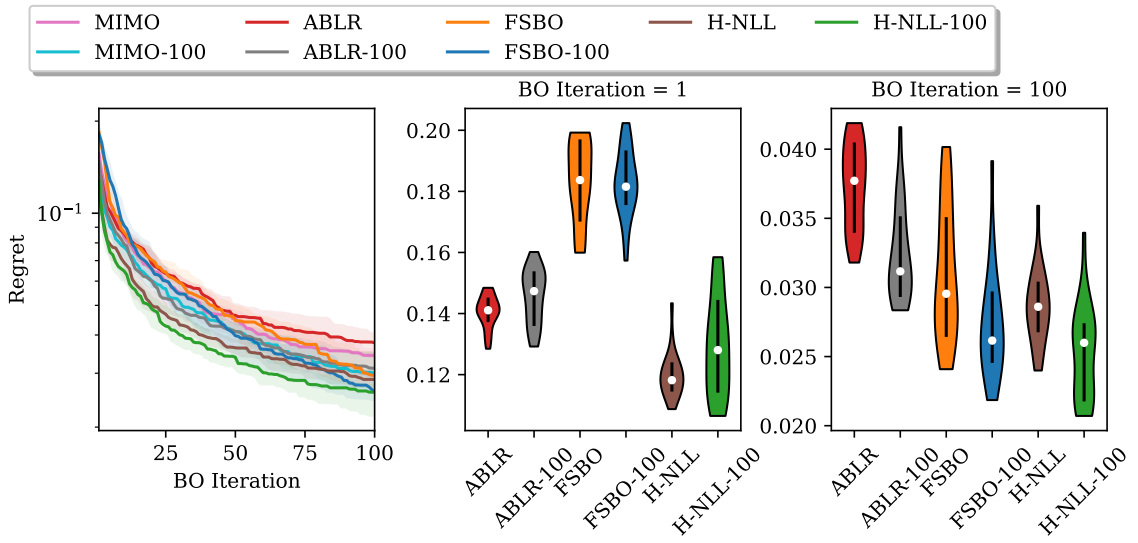

Figure 19: Regret curves to compare meta BO methods with at most 100 training datapoints per HPO-B training task (suffix "-100") versus all training data. We also show two vertical slices of the regret curves at the 1st and 100th iteration, where the white dot is the median and the black line is the 20/80 percentile. H-NLL-100 achieved top overall performance across BO iterations, surpassing H-NLL, which had access to the full training data.

7.6.3 THE "NEGATIVE TRANSFER" EFFECT IN HPO-B

"Negative transfer" refers to the situations where transfer learning negatively impacts the performance for test tasks (Rothfuss et al., 2021). This effect can be quantitatively shown by comparing the Training NLL and the regret. A positive correlation between the Training NLL and the regret means that optimizing the pre-training objective can positively contribute to the BO performance evaluated by regrets, which is the case for PD1 as shown in Figure 10. On the other hand, a negative correlation between the Training NLL and the regret indicates "negative transfer", which we observed in some search spaces of HPO-B.

We adopted the setup of §7.6.2 and pre-trained on at most 100 randomly sampled datapoints per training sub-dataset. Table 7 includes simple regrets at the 100th BO iteration for MIMO and HyperBO methods, as well as Training NLLs of HyperBO methods. We did not compute Test NLLs due to the large sizes of some test sub-datasets. While most HPO-B search spaces show positive correlations between the Training NLL and the regret, some search spaces, including 5636 and 5859, have negative Pearson correlations between Training NLLs and regrets. Within search spaces 5636, there are in fact 2 training functions (out of 54 in total) that are flat; i.e., all inputs have the same function values. For search space 5859, there are 4 flat training functions (out of 56 in total). In both of these two search spaces, none of the test functions are flat, suggesting that the flat training functions may negatively transfer the knowledge to test functions.

Figure 20 shows the aggregated results from search space 5636 and search space 5859. While the "negative transfer" effects are present in those two search spaces, H-NLL was still able to perform competitively and effectively lower the regrets for the test tasks in both search spaces.

	MIMO	ABLR		FSBO		H-NLL		
	Regret	Training NLL	Regret	Training NLL	Regret	Training NLL	Regret	Pearson
4796	21.62 ± 17.66	$\mathbf{-402 \pm 3}$	$\mathbf{1.73 \pm 0.25}$	-357 ± 3	$\mathbf{1.73 \pm 0.25}$	-381 ± 4	2.24 ± 0.25	0.05
5527	$\mathbf{26.29 \pm 6.77}$	$\mathbf{-342 \pm 9}$	27.43 ± 1.79	-314 ± 7	31.12 ± 1.35	-314 ± 10	33.40 ± 1.24	0.71
5636	114.56 ± 32.41	$\mathbf{-417 \pm 26}$	96.45 ± 3.86	-336 ± 13	49.39 ± 3.77	-356 ± 7	$\mathbf{26.50 \pm 2.09}$	-0.78
5859	54.41 ± 11.96	$\mathbf{-416 \pm 18}$	23.54 ± 2.45	-316 ± 16	$\mathbf{20.24 \pm 2.73}$	-342 ± 12	25.16 ± 1.36	-0.32
5860	12.29 ± 16.43	-221 ± 32	1.08 ± 0.21	-304 ± 21	$\mathbf{0.00 \pm 0.00}$	$\mathbf{-325 \pm 13}$	$\mathbf{0.00 \pm 0.00}$	0.88
5889	30.97 ± 31.84	$\mathbf{-308 \pm 1}$	41.32 ± 17.08	-301 ± 9	25.32 ± 4.99	-303 ± 11	$\mathbf{21.32 \pm 4.99}$	-0.33
5891	22.35 ± 1.93	-198 ± 9	30.59 ± 2.11	-223 ± 22	$\mathbf{19.89 \pm 0.65}$	$\mathbf{-237 \pm 24}$	20.16 ± 1.14	0.55
5906	$\mathbf{42.14 \pm 12.98}$	-273 ± 4	123.79 ± 13.16	-275 ± 7	74.78 ± 5.79	$\mathbf{-285 \pm 2}$	82.23 ± 10.43	0.38
5965	$\mathbf{28.43 \pm 3.06}$	$\mathbf{-387 \pm 4}$	34.27 ± 1.03	-298 ± 14	44.31 ± 2.20	-300 ± 40	42.20 ± 2.16	0.66
5970	3.05 ± 1.75	-124 ± 3	17.48 ± 8.12	-157 ± 7	4.63 ± 3.23	$\mathbf{-272 \pm 76}$	$\mathbf{1.17 \pm 0.31}$	0.53
5971	$\mathbf{3.15 \pm 1.28}$	-318 ± 3	4.75 ± 1.27	-321 ± 5	4.34 ± 0.51	$\mathbf{-322 \pm 4}$	3.85 ± 0.58	-0.03
6766	72.09 ± 5.79	-121 ± 2	75.96 ± 8.89	-159 ± 11	60.45 ± 9.66	$\mathbf{-254 \pm 77}$	$\mathbf{55.15 \pm 3.50}$	0.52
6767	$\mathbf{12.27 \pm 1.77}$	-317 ± 2	26.33 ± 3.55	-308 ± 13	32.00 ± 5.01	$\mathbf{-326 \pm 7}$	25.74 ± 7.65	0.43
6794	$\mathbf{31.02 \pm 4.73}$	$\mathbf{-449 \pm 3}$	33.05 ± 1.29	-428 ± 9	33.91 ± 1.05	-420 ± 29	36.36 ± 1.61	0.33
7607	$\mathbf{17.00 \pm 3.04}$	$\mathbf{-431 \pm 4}$	21.41 ± 2.75	-315 ± 30	33.99 ± 4.24	-337 ± 27	33.99 ± 1.90	0.69
7609	$\mathbf{4.13 \pm 0.62}$	$\mathbf{-425 \pm 10}$	5.61 ± 0.97	-304 ± 30	7.21 ± 1.02	-332 ± 18	7.77 ± 0.70	0.50

Table 7: The mean and standard deviation of Training NLLs (rounded to integers) and simple regrets ($\times 10^{-3}$) for each search space in HPO-B. We highlight the best Training NLL and simple regret in bold. Lower Training NLLs does not necessarily lead to lower regrets due to "negative transfer".

7.7 Summary of Experiments

Our experiments on PD1 and HPO-B showed the superior performance of HyperBO methods measured by simple regrets, fractions of succeeded tasks and rankings. H-NLL and H-EKL used the NLL and EKL loss functions defined in this work and overall performed better than competitive baselines. Our empirical analyses for PD1 showed evidence on the positive correlations between the NLL, EKL training losses, the NLL on test tasks and performance expressed by simple regrets. We studied the impact of reduced amount of training data and we also investigated the "negative transfer" problem in HPO-B. We found that HyperBO methods benefit from more training tasks and datapoints per task, and they are also robust to "negative transfer". Our results confirmed that pre-trained GPs can better describe the test functions and as a result, obtain better BO performance.

8. Discussion

Like most of the optimization literature, it is difficult to claim one approach is always better than the other. This phenomenon is widely recognized: "no free lunch in optimization". In this work, however, we showed that if we pre-train a Gaussian process (GP) and use it as the prior for Bayesian optimization (BO), we can escape "no free lunch" for a distribution of optimization tasks.

In the following, we discuss some interesting aspects of HyperBO. In §8.1, we present interpretations of HyperBO in a fully Bayesian manner. In §8.2, we identify potential extensions of HyperBO to enable more practical use cases. And finally in §8.3, we discuss unsolved open problems.

8.1 Fully Bayesian Interpretations of HyperBO

In §4, we assume the training functions $f_1, \cdots, f_N$ and the test function f are independently drawn from the same GP. This assumption is consistent with hierarchical Bayes settings, where all functions are independent conditioned on the GP. To construct fully Bayesian interpretations of HyperBO, one can place priors on the GP so that it is possible to perform complete Bayesian posterior updates on both the GP and the functions when we have more observed function values. There are different ways one can construct Bayesian hierarchical models for GPs. Here we focus on a generic hierarchical

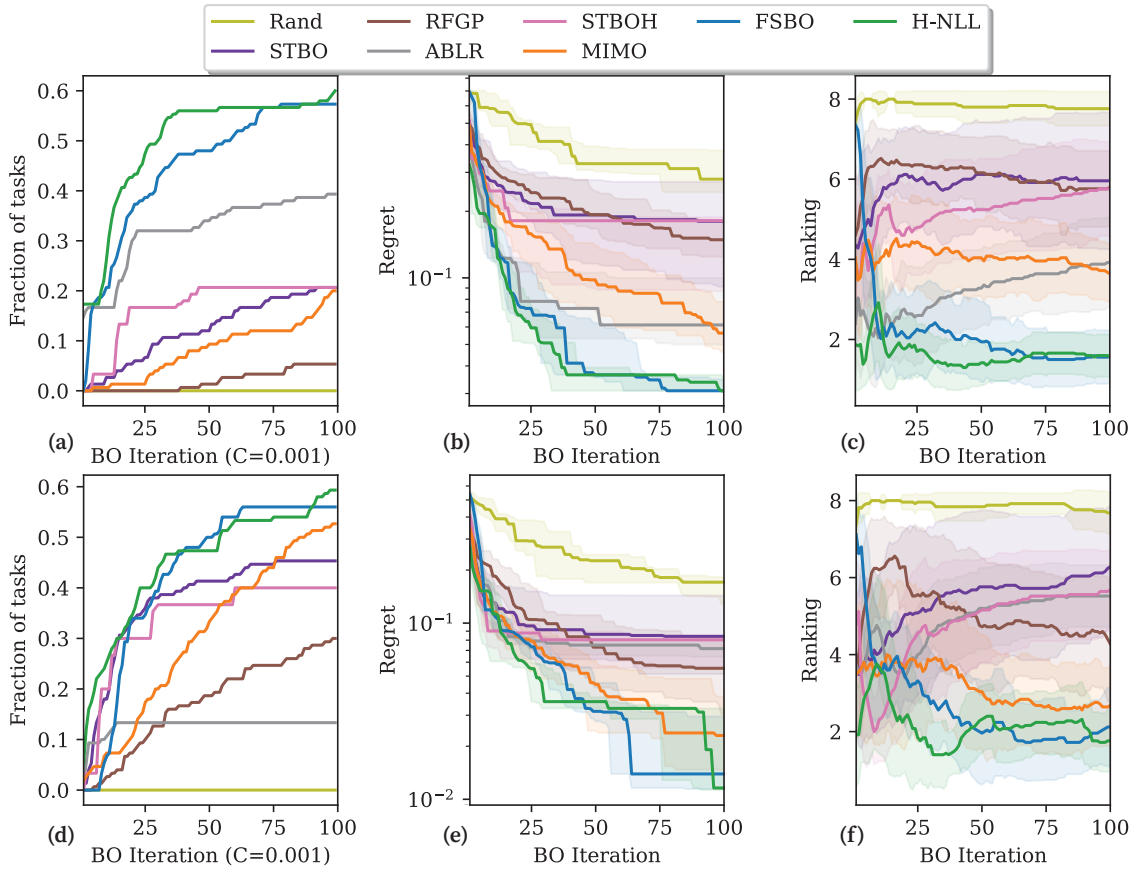

Figure 20: (a-c) are the performance profiles, regrets and ranking plots for search space 5636, and (d-f) are those for search space 5859. These search spaces contain flat training functions that encourage the "negative transfer" effect.

model where priors are placed on both the observation noise variance and the parameters of the mean and kernel functions.

More specifically, we assume that there is a parameter $\theta \sim p(\theta; \alpha)$; a mean function μ, a kernel function k and a noise variance parameter σ^2 are drawn from $p(\mu, k, \sigma^2 \mid \theta)$. Some functions $\{f_i\}_{i \in [N]}$ are then sampled independently from the same $\mathcal{GP}(\mu, k)$. The generative story of this hierarchical model is as follows:

- Draw parameter θ from $p(\theta; \alpha)$.
- Draw mean function μ, kernel function k and noise variance σ^2 from $p(\mu, k, \sigma^2 \mid \theta)$.
- For each function index i from 1 to N,
  - Draw a function f_i from $\mathcal{GP}(\mu, k)$.
  - For each datapoint index j from 1 to M_i,
    - Given input $x_j^{(i)}$, we draw the observation $y_j^{(i)} \sim \mathcal{N}\left(f_i(x_j^{(i)}), \sigma^2\right)$.

Figure 21 illustrates the graphical model for the hierarchical GP that we described above. We can simplify this hierarchical setting by restricting $p(\mu, k, \sigma^2 \mid \theta)$ to be Dirac delta functions: mean function μ, kernel function k and noise variance σ^2 are all fully determined given the parameter θ. This also means that the only free parameter in our model is θ. Now, given the training dataset $D_N = \{D_{f_i}\}_{i=1}^N$ and a sub-dataset D_f (the set of observations on the test function f), we can infer the posterior of the parameter θ: $p(\theta \mid D_N, D_f; \alpha)$. Then, we can also get the posterior for the test function f by marginalizing out the parameters

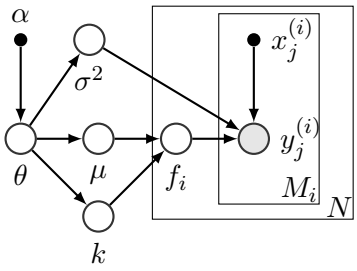

Figure 21: Graphical model for a hierarchical GP.

$$p(f \mid D_N, D_f; \alpha) = \int_\theta p(f \mid \theta) p(\theta \mid D_N, D_f; \alpha). \tag{26}$$

In other words, the posterior of function f depends on observations from all other conditionally *i.i.d.* function samples $f_1, \cdots, f_N$. So at each iteration of BO, we need to solve Eq. 26, which typically involves approximation and can have much worse time complexity (e.g., we may have to sample from the posterior over θ whenever we have new observations on f).

The solution we had in §5 can be interpreted as a cheap empirical Bayes approximation of Eq. 26: obtaining a point estimate for θ via MLE on training dataset D_N, and approximate the full posterior $p(f \mid D_N, D_f; \alpha)$ by removing D_N in the condition, i.e., $p(f \mid D_f, \theta)$.

Overall, it is indeed possible to interpret HyperBO in a fully Bayesian manner, but we found it not very meaningful as it does not provide new insights on methodology and brings us back to the original problem we identified: how to remove the known prior assumption in BO.

8.2 Important Extensions of HyperBO to Address Real-world Challenges of BO

In this work, we focused on the question of how to efficiently and effectively make use of multi-task training data to enable better BO via pre-training. Here we discuss extensions to our work that would enable even more flexible uses.

Batch evaluation. For simplicity of this paper, we did not consider batch evaluation but rather only focused on the prior selection dimension of the challenges in BO. However, it is straightforward to adopt any batch BO methods in conjunction with HyperBO to support obtaining observations in parallel. For example, we can directly use batch methods from Snoek et al. (2012); Kathuria et al. (2016); Wang et al. (2017b) etc. to replace line 5 of Algorithm 1.

High-dimensional and large scale data. Our hyperparameter tuning tasks are relatively low-dimensional (2-18 dimensions) and small scale BO problems. Because the tuning objectives are typically very expensive, we set the maximum number of BO observations to be 100. Other applications of BO might have high-dimensional inputs, and the number of observations for each BO problem might be much higher.

Our method can potentially be combined with high-dimensional and large scale BO methods to offer more capabilities. Those methods typically adopt probabilistic models different from vanilla GPs. In line 2 of Algorithm 1, we can adapt HyperBO to optimize the objective function in Eq. 3 for

the new model. Our meta-learning idea in this paper in fact also brings benefit to high-dimensional and large scale BO methods so that they can better identify their critical special structures, e.g., low-dimensional embeddings (Wang et al., 2016b), cylindrical kernels (Oh et al., 2018) or additive Mondrian kernels (Wang et al., 2018a).

Different search spaces. Roughly speaking, there could be two circumstances for different search spaces. Case I is that tasks share the same search variables, but the search ranges for some variables are different. For example, we may have each function $f_i : \mathfrak{X}_i \to \mathbb{R}, i \in [N]$ and $\mathfrak{X}_i = \prod_{j=1}^{d}[l_{ij}, h_{ij}] \subset \mathbb{R}^d$. In this case, our solution still applies by simply setting a union search space as $\mathfrak{X} = \bigcup_{i=1}^{N} \mathfrak{X}_i$ for learning and use the designated search space of new tasks for optimization.

Case II is more complicated: the search space for each function f_i is $\mathfrak{X}_i \subset \mathbb{R}^{d_i}$ and each dimension of $\mathfrak{X}_i$ may have a different meaning than another search space $\mathfrak{X}_j$ ($i \neq j$). This paper does not have an answer for this scenario, but concurrent works by Fan et al. (2022, 2024) provided some interesting solutions based on the HyperBO framework. Their key ideas are to use different setups of hierarchical GPs with stationary kernels and learn domain-specific priors or a universal prior for the shared parameters (e.g., lengthscales). Another concurrent work by Chen et al. (2022) solves Case II by pre-training a transformer to predict the next datapoints given a sequence of observations. This approach, however, may require millions of BO trajectories for training.

8.3 Other Open Problems

Along the way of developing HyperBO in this work, our findings and understandings have gradually revealed interesting open problems that are fundamental in hindsight but seem not to be well studied in the field. We list some of these problems below.

Bounded posterior predictions. Even if the pre-trained GP is very close to the ground truth GP, the posterior conditioned on the pre-trained GP may in fact deviate more and more from the ground truth posterior as we make more observations. Theorem 5 characterized this effect and suggested rescaling posterior for unbiased predictions of uncertainty. More studies are needed to better understand the assumptions and implications of Theorem 5. Can we use a different estimator to make the bounds tighter? Another potential question to ask is, if the loss in Eq. 1 (or the approximations in Eq. 5 or Eq. 13) is bounded, how does that bound translate to bounds on the posterior predictions of pre-trained GPs?

The necessary conditions for bounded regrets. As mentioned in §6.1, Eq. 19 (matching pre-trained posteriors with ground truth posteriors) and Eq. 20 (matching acquisition strategies) are sufficient but not necessary conditions to have bounded regrets. With Assumption 3 in §6.2.2, we demonstrated that we can show a near-zero regret bound as long as we can bound the differences of posterior mean and variance predictions (Theorem 5), which is another sufficient but not necessary condition. It is interesting to consider what the necessary conditions are for bounded regrets in HyperBO. If we know these necessary conditions, we will be able to know when HyperBO will fail.

Regret bounds for more practical settings. In practice, users of HyperBO might prefer to pre-train GPs with complex kernel and mean functions on continuous domains. However, in this paper, the regret bounds in Theorem 6 requires Assumption 3, which restricts the mean and kernels to be described by finite dimensional vectors and matrices. It remains unclear how to properly bound the regrets if Assumption 3 does not hold. Wang et al. (2018b) had a partial solution where we can relax Assumption 3 to have compact domain, but it required linear kernels and exact solutions to

linear systems, which is still not practical for real world settings. To fill the gap between theory and practice, we might need more guarantees on the optimization of the pre-training objectives.

As a side note, it is important not to forget that both the regret bounds and empirical studies serve the goal of verifying BO methods and identifying obvious pitfalls, so that we know how to use and adapt these methods in practice.

Invalid evaluations in real-world BO problems. As shown by our online experiments in Figure 14 in §7.5, the results can be very different from the offline ones. This is because hyperparameter evaluations can suffer from various conditions and eventually return infeasible values. For example, the training curves may diverge, hardware may have failures and certain numbers may exceed what can be represented in a finite number of bits (e.g., 64 bits for float). There is a rich literature on Bayesian optimization with constraints (Snoek, 2013; Gelbart et al., 2014; Gardner et al., 2014; Gelbart, 2015; Hernández-Lobato et al., 2016). However, many of the infeasible values we have observed do not fit well the assumptions made in this line of work. Specifically, the probability of receiving an infeasible value is not necessarily conditional on the inputs, or well modeled as a function from the inputs. Instead, phenomena such as hardware failures can occur in unstructured or structured ways that are difficult to model, such as being the result of a faulty GPU in a cluster. As such, this is perhaps better considered as a noise process rather than constraints on the input domain.

This scenario is not specific to machine learning hyperparameter tuning. For example, for control parameter tuning in robotics, the evaluation of a parameter depends on downstream modules (e.g., planning or control), and if any of the downstream modules fail, we cannot get a feasible evaluation. In our experiments in §7.5, we mapped the infeasible evaluations to a negative value, but this is not accurately reflecting the true evaluations if, for example, the hardware failed. In order to achieve high sample efficiency in BO for real-world problems, we need a more principled solution to account for such infeasible evaluations.

Pre-training other probabilistic models for function. Our pre-training method described in §5 is for GP priors, but it is not hard to see that any functional distribution could be pre-trained on multi-task data and the interpretation of the pre-trained prior is the distribution of the tasks. We may consider other kinds of prior models (Roininen et al., 2019; Tran et al., 2022; Gu et al., 2018; Pearce et al., 2022) that are more flexible than GPs for more complex datasets, but the challenge is in approximating the divergence between the ground truth and the model.

Alternative types of divergences as pre-training objectives. We used the KL divergence in our loss function in Eq. 1 of §5.1 to describe the "distance" between two GPs. However, as pointed out by Burt et al. (2020) and Sun et al. (2019), KL divergences between GPs can be ill-defined for GPs with parametric models. So if the pre-trained GP assigns measure zero to functions representable by the ground truth, the KL divergence would be infinite. There are other types of divergence or distance, e.g., Wasserstein distance, that may behave better, and there could be approximations that give posterior predictions that are more attuned to BO and other decision making tasks.

9. Conclusion

We proposed HyperBO: a framework that uses pre-training methods to learn a Gaussian process (GP) prior and subsequently perform Bayesian optimization (BO) with the learned prior. We developed KL divergence based pre-training methods and analyzed in detail the implications of using pre-trained GPs in BO. With HyperBO, we no longer have to hand-specify the exact quantitative parameters in a

GP prior. Instead, we only need to identify related tasks and past observations so that they can be used for pre-training.

Theoretically, we showed the regret bounds of HyperBO without the assumption that the prior is known. To verify HyperBO on real-world expensive hyperparameter tuning problems, we collected the PD1 Neural Net Tuning Dataset: the first hyperparameter tuning dataset for modern large-scale neural network model training. We demonstrated the superior empirical performance of HyperBO on both PD1 and other transfer learning BO benchmarks. Importantly, the idea of pre-training GPs brings a new perspective to interpret epistemic uncertainty in GPs, which are especially useful for Bayesian sequential decision making tasks exemplified by BO.

Acknowledgments

We would like to thank Zelda Mariet and Matthias Feurer for the help and consultation on transfer learning baselines. We would also like to thank Rif A. Saurous for constructive feedback, and Rodolphe Jenatton and David Belanger for feedback on previous versions of the manuscript. In addition, we thank Sharat Chikkerur, Ben Adlam, Balaji Lakshminarayanan, Fei Sha and Eytan Bakshy for comments.

Appendix A. Author Contributions

In the following, we list key contributions by authors.

- Zi Wang proposed the problem and designed the main HyperBO framework; set vision for the overall story of this work; conducted theoretical analyses; coordinated the design and completion of experiments, infrastructure and the new PD1 dataset; implemented the majority of HyperBO; conducted the experiments on HyperBO and some other baselines; consolidated discussions with co-authors and wrote the majority of the paper.

- George E. Dahl led the effort to create the PD1 Neural Net Tuning Dataset and co-led the init2winit project (infrastructure for PD1); managed all data collection experiments, launch scripts and data loading scripts with analyses; proposed to use performance profiles for experiment analyses; wrote §7.1; provided critical consultation on deep learning training algorithms and experiment design.

- Kevin Swersky refined HyperBO by adding implementations of input warping and prior regularizers; named HyperBO; fixed critical bugs of gradient computations in HyperBO; refined §3 to explain the principles behind HyperBO; significantly contributed to animated visualizations; contributed to the final revision and re-pivoting of this work.

- Chansoo Lee provided substantial support on the integration of HyperBO to internal toolboxes for online experiments; added essential internal baselines (STBOH, MIMO and RFGP) to experiments and wrote the descriptions and analyses; implemented the majority of GPax.

- Zachary Nado co-led the init2winit project (infrastructure for PD1) and designed new tasks for data collection; provided timely support on online experiments and integrated HyperBO into init2winit; provided consultation on deep learning training algorithms.

- Justin Gilmer co-led the init2winit project (infrastructure for PD1) and designed new tasks for data collection; supported internal infrastructure; provided consultation on deep learning training algorithms.

- Jasper Snoek advised on project direction, modeling choices, coding framework and paper writing; refined HyperBO by adding implementations of slice sampling, L-BFGS and warping priors; significantly contributed to code review; contributed to the final revision and re-pivoting of this work.

- Zoubin Ghahramani advised on project direction, philosophy, narratives and paper writing.

Appendix B. Derivations for EKL

In this section, we detail the derivations of EKL in §5.2.2 and its equivalence to NLL in special cases. To simplify the notations, we use p to denote the PDF of the ground truth multivariate Gaussian distribution, $\tilde{p}$ to denote the PDF of the MLE estimate with mean $\tilde{\boldsymbol{\mu}} \in \mathbb{R}^r$ and covariance $\tilde{\Sigma} \in \mathbb{R}^{r\times r}$, and q to denote the PDF of the model with mean $\boldsymbol{\mu} \in \mathbb{R}^r$ and covariance $\Sigma \in \mathbb{R}^{r\times r}$.

The KL divergence between two multivariate Gaussians. Assume that $\tilde{p}$ and q are non-degenerate. By definition, the KL divergence between $\tilde{p}$ and q is

$$\begin{aligned} D_{\mathrm{KL}}\Big(\tilde{p}, q\Big) &= H(\tilde{p}, q) - H(\tilde{p}) \\ &= \frac{1}{2}\mathbb{E}_{\boldsymbol{y}\sim\tilde{p}}\left[(\boldsymbol{y}-\boldsymbol{\mu})^\top\Sigma^{-1}(\boldsymbol{y}-\boldsymbol{\mu}) + \log|\Sigma| + r\log(2\pi)\right] - \frac{1}{2}\left(r\log(2\pi) + \log|\tilde{\Sigma}| + r\right) \\ &= \frac{1}{2}\left(\mathbb{E}_{\boldsymbol{y}\sim\tilde{p}}\left[(\boldsymbol{y}-\boldsymbol{\mu})^\top\Sigma^{-1}(\boldsymbol{y}-\boldsymbol{\mu})\right] + \log\frac{|\Sigma|}{|\tilde{\Sigma}|} - r\right). \end{aligned}$$

The expectation can be derived as follows,

$$\begin{aligned} \mathbb{E}_{\boldsymbol{y}\sim\tilde{p}}\left[(\boldsymbol{y}-\boldsymbol{\mu})^\top\Sigma^{-1}(\boldsymbol{y}-\boldsymbol{\mu})\right] &= \mathrm{tr}\left(\Sigma^{-1}\mathbb{E}_{\boldsymbol{y}\sim\tilde{p}}\left[(\boldsymbol{y}-\boldsymbol{\mu})(\boldsymbol{y}-\boldsymbol{\mu})^\top\right]\right) \\ &= \mathrm{tr}\left(\Sigma^{-1}\left(\mathbb{E}_{\boldsymbol{y}\sim\tilde{p}}\left[(\boldsymbol{y}-\tilde{\boldsymbol{\mu}})(\boldsymbol{y}-\tilde{\boldsymbol{\mu}})^\top\right] + (\tilde{\boldsymbol{\mu}}-\boldsymbol{\mu})(\tilde{\boldsymbol{\mu}}-\boldsymbol{\mu})^\top\right)\right) \\ &= \mathrm{tr}\left(\Sigma^{-1}\tilde{\Sigma}\right) + (\tilde{\boldsymbol{\mu}}-\boldsymbol{\mu})^\top\Sigma^{-1}(\tilde{\boldsymbol{\mu}}-\boldsymbol{\mu}). \end{aligned}$$

The derivation above uses the definition of the mean and covariance of Gaussian distributions, i.e., for $\tilde{p}$, its mean and covariance are defined as $\tilde{\boldsymbol{\mu}} = \mathbb{E}_{\boldsymbol{y}\sim\tilde{p}}[\boldsymbol{y}]$, $\tilde{\Sigma} = \mathbb{E}_{\boldsymbol{y}\sim\tilde{p}}\left[(\boldsymbol{y}-\tilde{\boldsymbol{\mu}})(\boldsymbol{y}-\tilde{\boldsymbol{\mu}})^\top\right]$.

Hence, we have

$$D_{\mathrm{KL}}\Big(\tilde{p}, q\Big) = \frac{1}{2}\left(\mathrm{tr}(\Sigma^{-1}\tilde{\Sigma}) + (\tilde{\boldsymbol{\mu}}-\boldsymbol{\mu})^\top\Sigma^{-1}(\tilde{\boldsymbol{\mu}}-\boldsymbol{\mu}) + \log\frac{|\Sigma|}{|\tilde{\Sigma}|} - r\right). \tag{27}$$

Eq. 27 can be used to derive Eq. 6 and it is consistent with Eq. 7.

The equivalence between EKL and NLL in special cases. Using the simplified notations, the claim in §5.5.1 is

$$D_{\mathrm{KL}}\Big(\tilde{p}, q\Big) \equiv -\frac{1}{N}\sum\nolimits_{i=1}^{N}\log q(\boldsymbol{y}_i) + \frac{1}{N}\sum\nolimits_{i=1}^{N}\log\tilde{p}(\boldsymbol{y}_i),$$

where $\boldsymbol{y}_i$ are *i.i.d.* samples from the ground truth p.

We prove the claim below.

By definition of the PDF of multivariate Gaussians, we have

$$\begin{aligned} -\frac{1}{N}\sum\nolimits_{i=1}^{N}\log\tilde{p}\left(\boldsymbol{y}_i\right) &= \frac{1}{2N}\sum\nolimits_{i=1}^{N}\left((\boldsymbol{y}_i-\tilde{\boldsymbol{\mu}})^\top\tilde{\Sigma}^{-1}(\boldsymbol{y}_i-\tilde{\boldsymbol{\mu}}) + \log|\tilde{\Sigma}| + r\log(2\pi)\right) \\ &= \frac{1}{2}\left(r\log(2\pi) + \log|\tilde{\Sigma}|\right) + \frac{1}{2N}\sum\nolimits_{i=1}^{N}(\boldsymbol{y}_i-\tilde{\boldsymbol{\mu}})^\top\tilde{\Sigma}^{-1}(\boldsymbol{y}_i-\tilde{\boldsymbol{\mu}}) \\ &= \frac{1}{2}\left(r\log(2\pi) + \log|\tilde{\Sigma}|\right) + \frac{1}{2}\mathrm{tr}\left(\tilde{\Sigma}^{-1}\tilde{\Sigma}\right) \\ &= \frac{1}{2}\left(r\log(2\pi) + \log|\tilde{\Sigma}| + r\right). \end{aligned}$$

The derivation uses the construction of MLE estimate for covariance matrix:

$$N\tilde{\Sigma} = \sum\nolimits_{i=1}^{N}\left(\boldsymbol{y}_i-\tilde{\boldsymbol{\mu}}\right)\left(\boldsymbol{y}_i-\tilde{\boldsymbol{\mu}}\right)^\top.$$

This means, $H(\tilde{p}) \equiv -\frac{1}{N}\sum_{i=1}^{N}\log\tilde{p}\left(\boldsymbol{y}_i\right)$.

Moreover, we have

$$\begin{aligned}-\frac{1}{N}\sum_{i=1}^{N}\log q\left(\boldsymbol{y}_i\right) &= \frac{1}{2N}\sum_{i=1}^{N}\Big((\boldsymbol{y}_i-\boldsymbol{\mu})^\top\Sigma^{-1}(\boldsymbol{y}_i-\boldsymbol{\mu})+\log|\Sigma|+r\log(2\pi)\Big)\\ &= \frac{1}{2}\left(r\log(2\pi)+\log|\Sigma|\right)+\frac{1}{2N}\sum_{i=1}^{N}(\boldsymbol{y}_i-\boldsymbol{\mu})^\top\Sigma^{-1}(\boldsymbol{y}_i-\boldsymbol{\mu}),\end{aligned}$$

and

$$\begin{aligned}\frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{y}_i-\boldsymbol{\mu})^\top\Sigma^{-1}(\boldsymbol{y}_i-\boldsymbol{\mu}) &= \text{tr}\left(\Sigma^{-1}\frac{1}{s}\sum_{i=1}^{N}(\boldsymbol{y}_i-\boldsymbol{\mu})(\boldsymbol{y}_i-\boldsymbol{\mu})^\top\right)\\ &= \text{tr}\left(\Sigma^{-1}\left(\frac{1}{N}\sum_{i=1}^{N}\left(\boldsymbol{y}_i-\tilde{\boldsymbol{\mu}}\right)\left(\boldsymbol{y}_i-\tilde{\boldsymbol{\mu}}\right)^\top+(\tilde{\boldsymbol{\mu}}-\boldsymbol{\mu})(\tilde{\boldsymbol{\mu}}-\boldsymbol{\mu})^\top\right)\right)\\ &= \text{tr}\left(\Sigma^{-1}\tilde{\Sigma}\right)+(\tilde{\boldsymbol{\mu}}-\boldsymbol{\mu})^\top\Sigma^{-1}(\tilde{\boldsymbol{\mu}}-\boldsymbol{\mu}).\end{aligned}$$

The derivation above for RHS uses the construction of MLE estimates for $\tilde{\boldsymbol{\mu}}$ and $\tilde{\Sigma}$ in Eq. 4, i.e.,

$$\tilde{\boldsymbol{\mu}}=\frac{1}{N}\sum_{i=1}^{N}\boldsymbol{y}_i,\qquad \frac{1}{N}\sum_{i=1}^{N}\boldsymbol{y}_i\tilde{\boldsymbol{\mu}}^\top=\tilde{\boldsymbol{\mu}}\tilde{\boldsymbol{\mu}}^\top,\qquad \tilde{\Sigma}=\frac{1}{N}\sum_{i=1}^{N}\left(\boldsymbol{y}_i-\tilde{\boldsymbol{\mu}}\right)\left(\boldsymbol{y}_i-\tilde{\boldsymbol{\mu}}\right)^\top.$$

By combining the equations above, we have

$$-\frac{1}{N}\sum_{i=1}^{N}\log q\left(\boldsymbol{y}_i\right)+\frac{1}{N}\sum_{i=1}^{N}\log\tilde{p}\left(\boldsymbol{y}_i\right)=\frac{1}{2}\left(\text{tr}(\Sigma^{-1}\tilde{\Sigma})+(\tilde{\boldsymbol{\mu}}-\boldsymbol{\mu})^\top\Sigma^{-1}(\tilde{\boldsymbol{\mu}}-\boldsymbol{\mu})+\log\frac{|\Sigma|}{|\tilde{\Sigma}|}-r\right),$$

which is equal to Eq. 27. Hence, the claim holds.

Appendix C. Proofs for Theorem 5 and Theorem 6

We provide proofs for Theorem 5, which bounds the posterior predictions of pre-trained Gaussian processes (GPs) and Theorem 6, which bounds the regrets of HyperBO. We also show an alternative regret bound (Theorem 18) in this section.

C.1 Theorem 5: Bounding Pre-trained GP Posterior Predictions

Recall that in Eq. 4, we used MLE to estimate the mean vector and covariance matrix of the ground truth GP on finite inputs. Proposition 4 then showed that the pre-trained GP on finite inputs is exactly the same as Eq. 4. So if we can bound posteriors based on Eq. 4, we can show Theorem 5.

Our plan for proving Theorem 5 is as follows. We first use the MLE estimates for a multivariate Gaussian distribution to derive a conditional distribution, and the parameters of that MLE based conditional distribution can be bounded by the parameters of the ground truth conditional distribution. This is done using the probability density functions for the MLE estimates of the mean vector and covariance matrix. Then, we can plug in our GP setups described in §6.2 and obtain Theorem 5.

The general proof strategy follows our prior work (Wang et al., 2018b), but in this work, we significantly simplified the proofs, improved readability and fixed issues with coefficients.

Assumption 7. *$\boldsymbol{y}_1,\cdots,\boldsymbol{y}_N\in\mathbb{R}^t$ are independent and identically distributed according to $\mathcal{N}(\boldsymbol{u},V)$, V is positive definite and $N>t$. Let $Y=[\boldsymbol{y}_i]_{i=1}^N\in\mathbb{R}^{t\times N}$. We denote the MLE estimators of $\boldsymbol{u},V$ as $\hat{\boldsymbol{u}}=\frac{1}{N}Y1_N\in\mathbb{R}^t$ and $\hat{V}=\frac{1}{N}(Y-\hat{\boldsymbol{u}}1_N^\top)(Y-\hat{\boldsymbol{u}}1_N^\top)^\top\in\mathbb{R}^{t\times t}$ where 1_N is a column vector of size N filled with 1s.*

Lemma 8. *Given Assumption 7, $\hat{u}$ and $\hat{V}$ are independent, and*

$$\hat{\boldsymbol{u}} \sim \mathcal{N}(\boldsymbol{u}, \frac{1}{N}V), \;\; \hat{V} \sim \mathcal{W}(\frac{1}{N}V, N-1).$$

Lemma 8 is a combination of Theorem 3.3.2 and Corollary 7.2.3 of Anderson (1958). Interested readers can find the proof of Lemma 8 in Anderson (1958).

Lemma 9. *For any $\hat{v} \sim \mathcal{W}(v, n), v \in \mathbb{R}$ and $b > 0$, we have*

$$\Pr[\frac{\hat{v}}{vn} \geq 1 + 2\sqrt{b} + 2b] \leq e^{-bn}, \;\; \Pr[\frac{\hat{v}}{vn} \leq 1 - 2\sqrt{b}] \leq e^{-bn}.$$

Proof. Given $\hat{v} \sim \mathcal{W}(v, n)$, $\frac{\hat{v}}{v}$ is distributed according to a chi-squared distribution with n degrees of freedom; namely, $\frac{\hat{v}}{v} \sim \chi^2(n)$. By Lemma 1 in Laurent and Massart (2000), we have

$$\Pr[\frac{\hat{v}}{v} - n \geq 2\sqrt{na} + 2a] \leq e^{-a}, \;\; \Pr[\frac{\hat{v}}{v} - n \leq -2\sqrt{na}] \leq e^{-a}.$$

Let $a = bn$, and we can reorganize the inequalities above as

$$\Pr[\frac{\hat{v}}{vn} \geq 1 + 2\sqrt{b} + 2b] \leq e^{-bn}, \;\; \Pr[\frac{\hat{v}}{vn} \leq 1 - 2\sqrt{b}] \leq e^{-bn}.$$

□

Lemma 10. *Assume Assumption 7 and $N > t+1$. For any $\boldsymbol{y} \sim \mathcal{N}(\boldsymbol{u}, V)$, we partition $\boldsymbol{y} = [y_\tau]_\tau^t \in \mathbb{R}^t$ to $\boldsymbol{y}_{t-1} = [y_\tau]_\tau^{t-1} \in \mathbb{R}^{t-1}$ and $y_t \in \mathbb{R}$; i.e., $\boldsymbol{y} = [\boldsymbol{y}_{t-1}, y_t]$. In the same way, we partition $\boldsymbol{u}$ and $\hat{\boldsymbol{u}}$. For matrix $V = [v_{\tau,\tau'}]_{\tau,\tau'=1}^t$, we partition it to $V_{t-1} = [v_{\tau,\tau'}]_{\tau,\tau'=1}^{t-1}$, $V_{t-1,t} = V_{t,t-1}^\top = [v_{\tau,t}]_\tau^t$ and v_t as a short-hand of $v_{t,t}$. We perform the same partition to matrix $\hat{V}$. That is,*

$$\boldsymbol{y} = \begin{bmatrix} \boldsymbol{y}_{t-1} \\ y_t \end{bmatrix}, \quad \boldsymbol{u} = \begin{bmatrix} \boldsymbol{u}_{t-1} \\ u_t \end{bmatrix}, \quad \hat{\boldsymbol{u}} = \begin{bmatrix} \hat{\boldsymbol{u}}_{t-1} \\ \hat{u}_t \end{bmatrix}, \quad V = \begin{bmatrix} V_{t-1} & V_{t-1,t} \\ V_{t,t-1} & v_t \end{bmatrix}, \quad \hat{V} = \begin{bmatrix} \hat{V}_{t-1} & \hat{V}_{t-1,t} \\ \hat{V}_{t,t-1} & \hat{v}_t \end{bmatrix}.$$

We also define the following short-hands ,

$$\omega = u_t + V_{t,t-1}V_{t-1}^{-1}(\boldsymbol{y}_{t-1} - \boldsymbol{u}_{t-1}), \;\; \kappa = v_t - V_{t,t-1}V_{t-1}^{-1}V_{t-1,t},$$
$$\hat{\omega} = \hat{u}_t + \hat{V}_{t,t-1}\hat{V}_{t-1}^{-1}(\boldsymbol{y}_{t-1} - \hat{\boldsymbol{u}}_{t-1}), \;\; \hat{\kappa} = \hat{v}_t - \hat{V}_{t,t-1}\hat{V}_{t-1}^{-1}\hat{V}_{t-1,t},$$

where $y_t \mid \boldsymbol{y}_{t-1} \sim \mathcal{N}(\omega, \kappa)$. Then, we have

$$\mathbb{E}[\hat{\omega}] = \omega, \;\; \mathbb{E}[\hat{\kappa}] = \frac{N-t}{N}\kappa. \tag{28}$$

Pick $\delta_1 \in (0,1)$ and $\delta_2 \in (0,1)$,

$$\Pr[|\hat{\omega} - \omega| < a\kappa] \geq 1 - \delta_1; \; a = \sqrt{\frac{2t + 4\sqrt{(t-1)\log\frac{2}{\delta_1}} + 4\log\frac{2}{\delta_1} - 4/N}{(N-t-1)\delta_1}}, \tag{29}$$

$$\Pr[1 - 2\sqrt{b} < \frac{N\hat{\kappa}}{(N-t)\kappa} < 1 + 2\sqrt{b} + 2b] > 1 - \delta_2; \;\; b = \frac{1}{N-t}\log\frac{2}{\delta_2}. \tag{30}$$

Proof. By Lemma 8 and Proposition 8.7 in Eaton (2007), we have

$$\hat{V} \sim \mathcal{W}(\frac{1}{N}V, N-1), \;\; \hat{\kappa} \sim \mathcal{W}(\frac{1}{N}\kappa, N-t).$$

Hence, $\mathbb{E}[\hat{\kappa}] = \frac{N-t}{N}\kappa$. By Lemma 9, we obtain

$$\Pr[\frac{N\hat{\kappa}}{(N-t)\kappa} \geq 1 + 2\sqrt{b} + 2b] \leq e^{-b(N-t)} = \frac{\delta_2}{2}, \;\; \Pr[\frac{N\hat{\kappa}}{(N-t)\kappa} \leq 1 - 2\sqrt{b}] \leq e^{-b(N-t)} = \frac{\delta_2}{2},$$

where $b = \frac{1}{N-t}\log\frac{2}{\delta_2}$. Thus, Eq. 30 holds. Next, we prove the claims about $\hat{\omega}$.

Our goal now is to bound the difference between $\hat{\omega}$ and ω. The plan is to first re-write random matrix $\hat{V}$ as a multiplication of two random matrices, then show the conditional distributions of terms involved in $\hat{\omega}$, and finally derive the expectation and variance of $\hat{\omega}$.

By Definition 8.1 of Eaton (2007), there exist $X = [\boldsymbol{x}_i]_{i=1}^{N-1} \in \mathbb{R}^{t\times(N-1)}, \boldsymbol{x}_i \overset{i.i.d.}{\sim} \mathcal{N}(0, \frac{1}{N}V)$ such that $\hat{V} = XX^\top$. Let $X = \left[\begin{smallmatrix} X_2 \\ X_1 \end{smallmatrix}\right]$, where $X_1 \in \mathbb{R}^{1\times(N-1)}$ (last row of X) and $X_2 \in \mathbb{R}^{(t-1)\times(N-1)}$. We then have $\hat{V}_{t-1} = X_2X_2^\top$, $\hat{V}_{t,t-1} = X_1X_2^\top$ and $\hat{v}_t = X_1X_1^\top$.

Conditional on X_2, since $\boldsymbol{x}_i \overset{i.i.d.}{\sim} \mathcal{N}(0, \frac{1}{N}V)$, elements of X_1 are independent, and each element has a Gaussian distribution conditioned on each column of X_2; that is,

$$X_1 \mid X_2 \sim \mathcal{N}(V_{t,t-1}(V_{t-1})^{-1}X_2, I_{N-1} \otimes \frac{1}{N}\kappa).$$

Hence, conditional on $\boldsymbol{y}_{t-1}, \hat{\boldsymbol{u}}_{t-1}$ and X_2, $\hat{\omega} - \hat{u}_t = X_1X_2^\top(X_2X_2^\top)^{-1}(\boldsymbol{y}_{t-1} - \hat{\boldsymbol{u}}_{t-1})$ is a weighted sum of *i.i.d.* random variables (elements of X_1).

Let $\bar{\boldsymbol{y}} = \boldsymbol{y}_{t-1} - \hat{\boldsymbol{u}}_{t-1}$. By applying the affine transformation $X_2^\top(X_2X_2^\top)^{-1}\bar{\boldsymbol{y}}$ to X_1, we have

$$\hat{\omega} - \hat{u}_t \mid \bar{\boldsymbol{y}}, X_2 \sim \mathcal{N}\left(V_{t,t-1}V_{t-1}^{-1}\bar{\boldsymbol{y}}, \; \frac{\kappa}{N}\bar{\boldsymbol{y}}^\top(X_2X_2^\top)^{-1}\bar{\boldsymbol{y}}\right).$$

The conditional distribution for $\hat{\omega} - \hat{u}_t$ depends on X_2 only through $\hat{V}_{t-1} = X_2X_2^\top$. So we can write it as

$$\hat{\omega} - \hat{u}_t \mid \bar{\boldsymbol{y}}, \hat{V}_{t-1} \sim \mathcal{N}\left(V_{t,t-1}V_{t-1}^{-1}\bar{\boldsymbol{y}}, \; \frac{\kappa}{N}\bar{\boldsymbol{y}}^\top\hat{V}_{t-1}^{-1}\bar{\boldsymbol{y}}\right).$$

Now we need to think about the distribution for $\hat{u}_t$. By Lemma 8, $\hat{\boldsymbol{u}}$ and $\hat{V}$ are independent. Hence $\hat{u}_t \perp \hat{V}_{t,t-1}$ and $\hat{u}_t \perp \hat{V}_{t-1}$. As a result, $\hat{u}_t \perp \hat{\omega} - \hat{u}_t \mid \hat{\boldsymbol{u}}_{t-1}$. Since $\hat{\boldsymbol{u}} \sim \mathcal{N}(\boldsymbol{u}, \frac{1}{N}V)$, we have

$$\hat{u}_t \mid \hat{\boldsymbol{u}}_{t-1} \sim \mathcal{N}\left(u_t + V_{t,t-1}V_{t-1}^{-1}(\hat{\boldsymbol{u}}_{t-1} - \boldsymbol{u}_{t-1})), \; \frac{\kappa}{N}\right).$$

We add $\hat{u}_t$ back to $\hat{\omega} - \hat{u}_t$, and get

$$\hat{\omega} \mid \boldsymbol{y}_{t-1}, \hat{\boldsymbol{u}}_{t-1}, \hat{V}_{t-1} \sim \mathcal{N}\left(\omega, \; \frac{\kappa}{N}(1 + \bar{\boldsymbol{y}}^\top\hat{V}_{t-1}^{-1}\bar{\boldsymbol{y}})\right).$$

By the law of total expectation, we can complete the proof for Eq. 28 as follows,

$$\mathbb{E}[\hat{\omega} \mid \boldsymbol{y}_{t-1}] = \mathbb{E}\left[\mathbb{E}[\hat{\omega} \mid \boldsymbol{y}_{t-1}, \hat{\boldsymbol{u}}_{t-1}, \hat{V}_{t-1}] \mid \boldsymbol{y}_{t-1}\right] = \omega.$$

By the law of total conditional variance,

$$\begin{aligned}\mathbb{V}[\hat{\omega} \mid \boldsymbol{y}_{t-1}] &= \mathbb{E}\left[\mathbb{V}[\hat{\omega} \mid \boldsymbol{y}_{t-1}, \hat{\boldsymbol{u}}_{t-1}, \hat{V}_{t-1}] \mid \boldsymbol{y}_{t-1}\right] + \mathbb{V}\left[\mathbb{E}[\hat{\omega} \mid \boldsymbol{y}_{t-1}, \hat{\boldsymbol{u}}_{t-1}, \hat{V}_{t-1}] \mid \boldsymbol{y}_{t-1}\right] \\ &= \mathbb{E}\left[\frac{\kappa}{N}(1 + \bar{\boldsymbol{y}}^\top \hat{V}_{t-1}^{-1} \bar{\boldsymbol{y}}) \mid \boldsymbol{y}_{t-1}\right] + \mathbb{V}\left[\omega \mid \boldsymbol{y}_{t-1}\right]\end{aligned}$$

Note that $\hat{V}_{t-1}^{-1}$ has an inverse Wishart distribution, i.e., $\hat{V}_{t-1}^{-1} \sim \mathcal{W}^{-1}(NV_{t-1}^{-1}, N-1)$; hence $\mathbb{E}[\hat{V}_{t-1}^{-1}] = \frac{N}{N-t-1} V_{t-1}^{-1}$. Since $\hat{\boldsymbol{u}}_{t-1} \sim \mathcal{N}(\boldsymbol{u}_{t-1}, \frac{1}{N} V_{t-1})$, we also have $N\bar{\boldsymbol{u}}^\top V_{t-1}^{-1} \bar{\boldsymbol{u}} \sim \chi^2(t-1)$ where $\bar{\boldsymbol{u}} = \hat{\boldsymbol{u}}_{t-1} - \boldsymbol{u}_{t-1}$ and $\chi^2(t-1)$ is the chi-squared distribution with $t-1$ degrees of freedom. Moreover, ω is a deterministic variable given $\boldsymbol{y}_{t-1}$. Hence,

$$\mathbb{V}[\hat{\omega} \mid \boldsymbol{y}_{t-1}] = \frac{1 + q - 2/N}{N - t - 1}\kappa,$$

where $q = (\boldsymbol{y}_{t-1} - \boldsymbol{u}_{t-1})^\top V_{t-1}^{-1} (\boldsymbol{y}_{t-1} - \boldsymbol{u}_{t-1})$.

Now we apply Chebyshev's inequality, and obtain

$$\Pr\left[|\hat{\omega} - \omega| < \sqrt{\frac{1 + q - 2/N}{(N - t - 1)\delta_1'}\kappa} \mid \boldsymbol{y}_{t-1}\right] \geq 1 - \delta_1'.$$

Notice that the randomness of q is from $\boldsymbol{y}_{t-1}$ and $\boldsymbol{y}_{t-1} \sim \mathcal{N}(\boldsymbol{u}_{t-1}, V_{t-1})$. This means $q \sim \chi^2(t-1)$. So we can further bound $q \leq t - 1 + 2\sqrt{(t-1)\log\frac{1}{\delta_1''}} + 2\log\frac{1}{\delta_1''}$ with probability at least $1 - \delta_1''$ by Lemma 1 in Laurent and Massart (2000). Hence, we can set $\delta_1' = \frac{\delta_1}{2}$ and $\delta_1'' = \frac{\delta_1}{2}$, and have

$$Pr\left[|\hat{\omega} - \omega| < a\sqrt{\kappa}\right] \geq 1 - \delta_1,$$

where $a = \sqrt{\frac{2t + 4\sqrt{(t-1)\log\frac{2}{\delta_1}} + 4\log\frac{2}{\delta_1} - 4/N}{(N-t-1)\delta_1}}$. Thus, Eq. 29 holds. □

Remarks. When bounding $\hat{\omega}$, we derived its mean and variance but did not try to characterize its distribution. In fact, we can write $\hat{\omega} = \omega_1 + \omega_2$, where

$$\begin{aligned}\omega_1 &\sim \mathcal{N}\left(\omega,\ \frac{\kappa}{N}\right) \\ \omega_2 &\sim \mathcal{N}\left(0,\ \frac{\kappa}{N}(\boldsymbol{y}_{t-1} - \hat{\boldsymbol{u}}_{t-1})^\top \hat{V}_{t-1}^{-1} (\boldsymbol{y}_{t-1} - \hat{\boldsymbol{u}}_{t-1})\right).\end{aligned}$$

Let $\omega_2 = \sqrt{\kappa}(\boldsymbol{y}_{t-1} - \hat{\boldsymbol{u}}_{t-1})^\top \omega_3$, where $\hat{\boldsymbol{u}}_{t-1} \sim \mathcal{N}(\boldsymbol{u}_{t-1}, \frac{1}{N} V_{t-1})$ and $\omega_3 \in \mathbb{R}^{t-1}$. Then we have

$$\omega_3 \sim \mathcal{N}\left(0,\ \frac{1}{N}\hat{V}_{t-1}^{-1}\right);\ \ \hat{V}_{t-1}^{-1} \sim \mathcal{W}^{-1}(NV_{t-1}^{-1}, N-1),$$

which means $(\omega_3, \hat{V}_{t-1}^{-1})$ is distributed according to a normal-inverse-Wishart distribution. Interested readers can derive a tighter bound for $\hat{\omega}$ based on this decomposition.

Proof of Theorem 5. The following proof is for Theorem 5 in §6.2.1.

Proof. (Theorem 5) Recall that in the $t-$th iteration at line 5 of Algorithm 1, we have observations $D_f = \{(x_\tau, y_\tau)\}_{\tau=1}^{t-1}$. Let $\boldsymbol{x} = [x_\tau]_{\tau=1}^{t-1}$ and $\boldsymbol{y}' = [y_\tau]_{\tau=1}^{t-1}$.

The ground truth posterior is $\mathcal{GP}(\mu^*, k^* \circ \sigma_*^2 \mid D_f) := \mathcal{GP}(\mu_{t-1}^*, k_{t-1}^*)$, where

$$\mu_{t-1}^*(x) = \mu^*(x) + k^*(x, \boldsymbol{x})(k^* \circ \sigma_*^2(\boldsymbol{x}))^{-1}(\boldsymbol{y}' - \mu^*(\boldsymbol{x})), \ \ \forall x \in \mathfrak{X},$$
$$k_{t-1}^*(x, x') = k^*(x, x') - k^*(x, \boldsymbol{x})(k^* \circ \sigma_*^2(\boldsymbol{x}))^{-1}k^*(\boldsymbol{x}, x'), \ \ \forall x, x' \in \mathfrak{X}.$$

The pre-trained GP posterior is $\mathcal{GP}(\hat{\mu}, \hat{k} \circ \hat{\sigma}^2 \mid D_f) := \mathcal{GP}(\hat{\mu}_{t-1}, \hat{k}_{t-1})$, where

$$\hat{\mu}_{t-1}(x) := \hat{\mu}(x) + \hat{k}(x, \boldsymbol{x})(\hat{k} \circ \hat{\sigma}^2(\boldsymbol{x}))^{-1}(\boldsymbol{y}' - \hat{\mu}(\boldsymbol{x})), \ \ \forall x \in \mathfrak{X},$$
$$\hat{k}_{t-1}(x, x') := \hat{k}(x, x') - \hat{k}(x, \boldsymbol{x})(\hat{k} \circ \hat{\sigma}^2(\boldsymbol{x}))^{-1}\hat{k}(\boldsymbol{x}, x'), \ \ \forall x, x' \in \mathfrak{X}.$$

We plug in the following terms to Lemma 10.

$$\boldsymbol{y} \leftarrow \begin{bmatrix} \boldsymbol{y}' \\ f(x) \end{bmatrix}, \quad \boldsymbol{u} \leftarrow \begin{bmatrix} \mu^*(\boldsymbol{x}) \\ \mu^*(x) \end{bmatrix}, \quad \hat{\boldsymbol{u}} \leftarrow \begin{bmatrix} \hat{\mu}(\boldsymbol{x}) \\ \hat{\mu}(x) \end{bmatrix},$$
$$V \leftarrow \begin{bmatrix} k^* \circ \sigma_*^2(\boldsymbol{x}) & k^* \circ \sigma_*^2(\boldsymbol{x}, x) \\ k^* \circ \sigma_*^2(x, \boldsymbol{x}) & k^* \circ \sigma_*^2(x) \end{bmatrix}, \quad \hat{V} \leftarrow \begin{bmatrix} \hat{k} \circ \hat{\sigma}^2(\boldsymbol{x}) & \hat{k} \circ \hat{\sigma}^2(\boldsymbol{x}, x) \\ \hat{k} \circ \hat{\sigma}^2(x, \boldsymbol{x}) & \hat{k} \circ \hat{\sigma}^2(x) \end{bmatrix}.$$

Hence,

$$\omega = \mu_{t-1}^*(x), \ \ \kappa = k_{t-1}^*(x) + \sigma_*^2 = k_{t-1}^* \circ \sigma_*^2(x),$$
$$\hat{\omega} = \hat{\mu}_{t-1}(x), \ \ \hat{\kappa} = \hat{k}_{t-1}(x) + \hat{\sigma}^2 = \hat{k}_{t-1} \circ \hat{\sigma}^2(x),$$

Furthermore, we set $\delta_1 \leftarrow \delta/2$ and $\delta_2 \leftarrow \delta/2$. Thus, with probability at least $1 - \delta$, we have

$$|\hat{\mu}_{t-1}(x) - \mu_{t-1}^*(x)|^2 < a^2 k_{t-1}^* \circ \sigma_*^2(x) \text{ and } 1 - 2\sqrt{b} < \frac{N\hat{k}_{t-1} \circ \hat{\sigma}^2(x)}{(N-t)k_{t-1}^* \circ \sigma_*^2(x)} < 1 + 2\sqrt{b} + 2b,$$

where $a^2 = \frac{4\left(t + 2\sqrt{(t-1)\log\frac{4}{\delta}} + 2\log\frac{4}{\delta} - 2/N\right)}{(N-t-1)\delta}, b = \frac{1}{N-t}\log\frac{4}{\delta}$. □

This completes the proof for Theorem 5.

C.2 Theorem 6: Bounding the Regrets of HyperBO

Theorem 6 is about regret bounds of HyperBO with acquisition strategies that are special cases of GP-UCB and PI. Recall that in §4, we defined simple regret as our metric: $r_T = \max_{x \in \mathfrak{X}} f(x) - f(\hat{x})$, and we set $\hat{x} = x_\tau, \tau = \arg\max_{t \in [T]} y_t$. We need to bound r_T for running HyperBO on a test function $f \sim \mathcal{GP}(\mu^*, k^* \circ \sigma_*^2)$. Note that HyperBO does not have access to the ground truth $\mathcal{GP}(\mu^*, k^* \circ \sigma_*^2)$. Hence classic theoretical analyses tools such as Srinivas et al. (2010) do not apply to HyperBO.

In Theorem 5, we showed that the pre-trained GP posterior $\mathcal{GP}(\hat{\mu}, \hat{k} \circ \hat{\sigma}^2 \mid D_f)$ can be close to the ground truth posterior $\mathcal{GP}(\mu^*, k^* \circ \sigma_*^2 \mid D_f)$. So the hope is that optimizing an acquisition function defined by $\mathcal{GP}(\hat{\mu}, \hat{k} \circ \hat{\sigma}^2 \mid D_f)$ can get similar results to optimizing an acquisition function defined by $\mathcal{GP}(\mu^*, k^* \circ \sigma_*^2 \mid D_f)$. However, these two posteriors are still different, and we need to make sure the acquisition functions are informed of the difference.

We analyze the following two acquisition functions.

- GP-UCB (Srinivas et al., 2010) with explore-exploit trade-off parameter β:

$$\alpha_{t-1}^{\text{UCB}}(x) = \hat{\mu}_{t-1}(x) + \beta\sqrt{\hat{k}_{t-1} \circ \hat{\sigma}^2(x)}, \tag{31}$$

  where β is defined by a, b from Theorem 5.

- The max-value version of PI (Wang and Jegelka, 2017; Kushner, 1964) with $\hat{f}^* \geq \max_{x\in\mathfrak{X}} f(x)$:

$$\alpha_{t-1}^{\text{PI}}(x) = \frac{\hat{\mu}_{t-1}(x) - \hat{f}^*}{\sqrt{\hat{k}_{t-1} \circ \hat{\sigma}^2(x)}}. \tag{32}$$

Taking GP-UCB as an example, the proof strategy is to write r_T with the pre-trained GP posterior mean $\hat{\mu}_{t-1}$ and kernel $\hat{k}_{t-1}(x)$. This is to ensure that we can remove the regret's dependency on μ_{t-1} by using the critical information that for any iteration $t = 1, \cdots, T$, $\alpha_{t-1}^{\text{UCB}}(x_t) \geq \alpha_{t-1}^{\text{UCB}}(x^*)$, where x_t is the maximizer of the acquisition function and x^* is the maximizer of test function f. Once r_T is written with the pre-trained GP posterior, we can use Theorem 5 to bound $\hat{\mu}_{t-1}, \hat{k}_{t-1}$ in r_T with the ground truth $k_{t-1}^*(x_t)$. Then, we can use the max information gain $\max_{D_f} I(f; D_f)$ to bound $k_{t-1}^*(x_t)$. Since r_T can be fully expressed by $k_{t-1}^*(x_t)$, we can conclude the regret bound for r_T with GP-UCB. The case for PI is similar.

Next, we review some useful lemmas before proceeding to the proofs related to Theorem 6.

Corollary 11 (Bernoulli's inequality)**.** *For all $0 \leq x \leq c$ and $a > 0$, we have $x \leq \frac{c\log(1+\frac{ax}{c})}{\log(1+a)}$.*

Proof. By Bernoulli's inequality, $(1+a)^{\frac{x}{c}} \leq 1 + \frac{ax}{c}$. Because $\log(1+a) > 0$, by rearranging, we have $x \leq \frac{c\log(1+\frac{ax}{c})}{\log(1+a)}$. □

Lemma 12. *For any $0 \leq x \leq c$ and $a > 0$, we have $\sqrt{x} < \sqrt{x+a} - \frac{a}{2\sqrt{c+a}}$.*

Proof. Numerically, for any $n \geq 1$, $\frac{1}{\sqrt{n}} < 2\sqrt{n} - 2\sqrt{n-1}$ (Weisstein, 1999-2018). Let $n = \frac{x}{a} + 1$. Then, we have

$$\begin{aligned}
\frac{1}{\sqrt{\frac{x}{a}+1}} &< 2\sqrt{\frac{x}{a}+1} - 2\sqrt{\frac{x}{a}} \\
\frac{a}{\sqrt{a+c}} < \frac{a}{\sqrt{a+x}} &< 2\sqrt{x+a} - 2\sqrt{x} \\
\sqrt{x} &< \sqrt{x+a} - \frac{a}{2\sqrt{a+c}}.
\end{aligned}$$

□

Lemma 13 (Lemma 5.3 of Srinivas et al. (2010))**.** *Let $\boldsymbol{x}_T = [x_t]_{t=1}^T \subseteq \mathfrak{X}$ and $f \sim \mathcal{GP}(\mu, k)$ with observation noise σ^2. The mutual information between the function values $f(\boldsymbol{x}_T)$ and their observations $\boldsymbol{y}_T = [y_t]_{t=1}^T$ satisfies*

$$I(f(\boldsymbol{x}_T); \boldsymbol{y}_T) = \frac{1}{2}\sum_{t=1}^T \log(1 + \sigma^{-2}k_{t-1}(x_t)).$$

Corollary 14 (Srinivas et al. (2010))**.** *For any Gaussian variable $x \sim \mathcal{N}(\mu, \sigma^2), x \in \mathbb{R}$,*

$$\Pr[x - \mu \leq \zeta\sigma] \geq 1 - \delta,\ \Pr[x - \mu \geq -\zeta\sigma] \geq 1 - \delta$$

where $\zeta = (2\log(\frac{1}{2\delta}))^{\frac{1}{2}}$ and $\delta \in (0, 1)$.

Proof. Let $z = \frac{\mu - x}{\sigma} \sim \mathcal{N}(0,1)$. We have

$$\begin{aligned}
\Pr[z > \zeta] &= \int_{\zeta}^{+\infty} \frac{1}{\sqrt{2\pi}} e^{-z^2/2} \, \mathrm{d}z \\
&= \int_{\zeta}^{+\infty} \frac{1}{\sqrt{2\pi}} e^{-(z-\zeta)^2/2 - \zeta^2/2 - z\zeta} \, \mathrm{d}z \\
&\leq e^{-\zeta^2/2} \int_{\zeta}^{+\infty} \frac{1}{\sqrt{2\pi}} e^{-(z-\zeta)^2/2} \, \mathrm{d}z \\
&= \frac{1}{2} e^{-\zeta^2/2},
\end{aligned}$$

since $e^{-z\zeta} \leq 1$ with $z \geq \zeta$. Similarly, $\Pr[z < -\zeta] \leq \frac{1}{2} e^{-\zeta^2/2}$. We reach the conclusion by rearranging the constants. □

The next lemma bounds the difference between the best sample in observations and the best observed sample.

Lemma 15. *With probability at least* $1 - \delta$*, we have*

$$f(x_{\tau'}) - f(x_\tau) \leq 2(2 \log \frac{1}{\delta})^{\frac{1}{2}} \sigma,$$

where $\tau' = \arg\max_{t \in [T]} f(x_t)$ *and* $\tau = \arg\max_{t \in [T]} y_t$.

Proof. Note that $y_t \sim \mathcal{N}(f(x_t), \sigma^2), \forall t \in [T]$. By Corollary 14, with probability at least $1 - \delta$,

$$f(x_\tau) + C\sigma \geq y_\tau \geq y_{\tau'} \geq f(x_{\tau'}) - C\sigma,$$

where $C = (2 \log \frac{1}{\delta})^{\frac{1}{2}}$. Hence $f(x_{\tau'}) - f(x_\tau) \leq 2C\sigma$. □

Now we are fully equipped to prove the regret bounds. Next, we show the regret bound for GP-UCB (Eq. 31) in Theorem 16, and then PI (Eq. 32) in Theorem 17. Note that we do not assume the ground truth GP is known, but we use the ground truth GP to bound the regret. We provide an alternative regret bound parameterized by the pre-trained GP in §C.3.

Theorem 16. *Assume* $N > T + 4 \log \frac{12}{\delta}$*,* $\delta \in (0,1)$*, and there exists constant* c *such that* $c \geq k^*(x)$ *for any* $x \in \mathfrak{X}$*. With probability at least* $1 - \delta$*, GP-UCB in Eq. 31 obtains the simple regret*

$$r_T^{\textit{GP-UCB}} \leq (a + \zeta)(1 + s) \sqrt{\frac{2c\rho_T}{T \log(1 + c\sigma_*^{-2})} + \sigma_*^2} + \zeta\sigma_*(1 - \frac{\sigma_*}{\sqrt{c + \sigma_*^2}}),$$

where $\beta = (a + \zeta)\sqrt{\frac{N}{(N-T)(1-2\sqrt{b})}}$*,* $\zeta = \sqrt{2 \log(\frac{3}{\delta})}$*,* $a^2 = \frac{12\left(T + 2\sqrt{(T-1)\log\frac{12}{\delta}} + 2\log\frac{12}{\delta} - 2/N\right)}{(N-T-1)\delta}$*,* $b = \frac{1}{N-T} \log \frac{12}{\delta}$*,* $s = \sqrt{\frac{1+2\sqrt{b}+2b}{1-2\sqrt{b}}}$*, and* $\rho_T = \max_{A \subseteq \mathfrak{X}, |A| = T} \frac{1}{2} \log |\boldsymbol{I} + \sigma^{-2} k^*(A)|$.

Proof. Let $\tau = \arg\min_{t\in[T]} k^*_{t-1}(x_t)$, $x^* = \arg\max_{x\in\mathfrak{X}} f(x)$ and $\hat{x} = x_{t'}$ where $t' = \arg\max_{t\in[T]} y_t$. We set $\zeta = \sqrt{2\log(\frac{1}{\delta'})}$ with $\delta' = \delta/3$. By Lemma 15 , with probability at least $1-\delta'$,

$$\begin{aligned} r_T^{\text{GP-UCB}} &= \big(f(x^*) - \max_{t\in[T]} f(x_t)\big) + \big(\max_{t\in[T]} f(x_t) - f(\hat{x})\big) \\ &\le r + \zeta\sigma, \end{aligned}$$

where $r := f(x^*) - \max_{t\in[T]} f(x_t)$. We use the following short-hands for simplicity:

$$\begin{aligned} &\Delta_\mu = \mu^*_{\tau-1}(x^*) - \mu^*_{\tau-1}(x_\tau), \quad \hat{\Delta}_\mu = \hat{\mu}_{\tau-1}(x^*) - \hat{\mu}_{\tau-1}(x_\tau), \\ &\lambda(\cdot) = \sqrt{k^*_{\tau-1}(\cdot) + \sigma_*^2}, \quad \hat{\lambda}(\cdot) = \sqrt{\hat{k}_{\tau-1}(\cdot) + \hat{\sigma}^2}, \quad \text{and} \ \ \psi(\cdot) = \sqrt{k^*_{\tau-1}(\cdot)} \end{aligned}$$

Next we bound r. By Corollary 14, with probability at least $1-\delta'$,

$$\begin{aligned} r &\le f(x^*) - f(x_\tau) \\ &\le \Delta_\mu + \zeta\psi(x^*) + \zeta\psi(x_\tau). \end{aligned}$$

By Theorem 5 and that $\tau \le T$, the following inequality holds with probability at least $1-\delta'$,

$$\Delta_\mu < \hat{\Delta}_\mu + a\lambda(x^*) + a\lambda(x_\tau) \ \text{ and } \ \hat{\lambda}(x_\tau) < h\lambda(x_\tau) \ \text{ and } \ \lambda(x^*) < h'\hat{\lambda}(x^*).$$

where $a^2 = \frac{4\left(T+2\sqrt{(T-1)\log\frac{4}{\delta'}}+2\log\frac{4}{\delta'}-2/N\right)}{(N-T-1)\delta'}, b = \frac{1}{N-T}\log\frac{4}{\delta'}, h = \sqrt{\frac{(1+2\sqrt{b}+2b)(N-T)}{N}}, h' = \sqrt{\frac{N}{(N-T)(1-2\sqrt{b})}}$.

By Lemma 12 and that $c \ge \sup_{x\in\mathfrak{X}} k^*(x) \ge k^*_{\tau-1}(x^*)$, we have,

$$\psi(x^*) \le \lambda(x^*) - \frac{\sigma_*^2}{2\sqrt{c+\sigma_*^2}}$$

Notice that because of the acquisition strategy of GP-UCB, i.e., $\alpha^{\text{UCB}}_{\tau-1}(x) = \hat{\mu}_{\tau-1}(x) + \beta\hat{\lambda}(x)$, the following inequality holds,

$$\begin{aligned} \hat{\mu}_{\tau-1}(x^*) + \beta\hat{\lambda}(x^*) \le \hat{\mu}_{\tau-1}(x_\tau) + \beta\hat{\lambda}(x_\tau), \quad \text{i.e.,} \\ \hat{\Delta}_\mu \le \beta\left(\hat{\lambda}(x_\tau) - \hat{\lambda}(x^*)\right). \end{aligned}$$

Note that $\delta' = \delta/3$, $\beta = (a+\zeta)h'$ and $\sqrt{\sigma_1^2+\sigma_2^2} < \sigma_1+\sigma_2$ for $\sigma_1, \sigma_2 > 0$. And let $s = hh' = \sqrt{\frac{1+2\sqrt{b}+2b}{1-2\sqrt{b}}}$. Hence, with probability at least $1-\delta$,

$$\begin{aligned} r &< \Delta_\mu + \zeta\lambda(x^*) + \zeta\psi(x_\tau) - \frac{\zeta\sigma_*^2}{2\sqrt{c+\sigma_*^2}} \\ &< \hat{\Delta}_\mu + (a+\zeta)\lambda(x^*) + a\lambda(x_\tau) + \zeta\psi(x_\tau) - \frac{\zeta\sigma_*^2}{2\sqrt{c+\sigma_*^2}} \\ &< \beta\left(\hat{\lambda}(x_\tau) - \hat{\lambda}(x^*)\right) + (a+\zeta)h'\hat{\lambda}(x^*) + a\lambda(x_\tau) + \zeta\psi(x_\tau) - \frac{\zeta\sigma_*^2}{2\sqrt{c+\sigma_*^2}} \\ &< ((a+\zeta)s + a + \zeta)\lambda(x_\tau) \ - \frac{\zeta\sigma_*^2}{\sqrt{c+\sigma_*^2}} \end{aligned}$$

By Corollary 11 and the fact that $\tau = \arg\min_{t\in[T]} k^*_{t-1}(x_t)$, we have

$$\begin{aligned} k^*_{\tau-1}(x_\tau) &\le \frac{1}{T}\sum_{t=1}^{T} k^*_{t-1}(x_t) \\ &\le \frac{1}{T}\sum_{t=1}^{T} \frac{c\log(1+\frac{c\sigma_*^{-2}k^*_{t-1}(x_t)}{c})}{\log(1+c\sigma_*^{-2})} \\ &= \frac{c}{T\log(1+c\sigma_*^{-2})}\sum_{t=1}^{T}\log(1+\sigma_*^{-2}k^*_{t-1}(x_t)). \end{aligned}$$

Notice that here Corollary 11 applies because $0 \le k^*_{\tau-1}(x_\tau) \le c$.

By Lemma 13, we have $I(f(\boldsymbol{x}_T);\boldsymbol{y}_T) = \frac{1}{2}\sum_{t=1}^{T}\log(1+\sigma_*^{-2}k^*_{t-1}(x_t)) \le \rho_T$, so

$$k^*_{\tau-1}(x_\tau) \le \frac{2c\rho_T}{T\log(1+c\sigma_*^{-2})},$$

which implies

$$r_T^{\text{GP-UCB}} < (a+\zeta)(1+s)\sqrt{\frac{2c\rho_T}{T\log(1+c\sigma_*^{-2})}+\sigma_*^2} + \zeta\sigma_*(1-\frac{\sigma_*}{\sqrt{c+\sigma_*^2}}).$$

Note that ρ_T is the max information gain over T observations of the ground truth GP. □

Theorem 17. *Assume $N > T + 4\log\frac{12}{\delta}$, $\delta \in (0,1)$, and there exists constant $c = \sup_{x\in\mathfrak{X}} k^*(x)$. With probability at least $1-\delta$, the simple regret in T iterations of PI in Eq. 32 that uses $\hat{f}^* \ge \max_{x\in\mathfrak{X}} f(x)$ as its target value satisfies*

$$r_T^{PI} < \eta\sqrt{\frac{2c\rho_T}{T\log(1+c\sigma_*^{-2})}+\sigma_*^2} + \zeta\sigma_*(1-\frac{\sigma_*}{2\sqrt{c+\sigma_*^2}}),$$

where $\eta = s(\frac{\hat{f}^*-\mu^*_{\tau-1}(x^*)}{\sqrt{k^*_{\tau-1}(x^*)+\sigma_*^2}}+a)+a+\zeta$, *and* $\zeta = \sqrt{2\log(\frac{3}{\delta})}$, $a^2 = \frac{12\left(T+2\sqrt{(T-1)\log\frac{12}{\delta}}+2\log\frac{12}{\delta}-2/N\right)}{(N-T-1)\delta}$, $b = \frac{1}{N-T}\log\frac{12}{\delta}$, $\rho_T = \max_{A\subseteq\mathfrak{X},|A|=T}\frac{1}{2}\log|\boldsymbol{I}+\sigma^{-2}\hat{k}(A)|$, $s = \sqrt{\frac{1+2\sqrt{b}+2b}{1-2\sqrt{b}}}$, $\tau = \arg\min_{t\in[T]} k^*_{t-1}(x_t)$.

Proof. The acquisition strategy PI in Eq. 32 uses target value $\hat{f}^*$ which is an upper bound on f. By Corollary 14, with probability at least $1-\delta'$,

$$\begin{aligned} \hat{f}^* - \max_{t\in[T]} f(x_t) &\le \hat{f}^* - f(x_\tau) \\ &\le \hat{f}^* - \mu^*_{\tau-1}(x_\tau) + \mu^*_{\tau-1}(x_\tau) - f(x_\tau) \\ &\le \hat{f}^* - \mu^*_{\tau-1}(x_\tau) + \zeta\sqrt{k^*_{\tau-1}(x_\tau)}, \end{aligned}$$

where $\zeta = \sqrt{2\log(\frac{1}{\delta'})}$ and $\tau = \arg\min_{t\in[T]} k^*_{t-1}(x_t)$. We use the following short-hands for simplicity:

$$\lambda_\tau = \sqrt{k^*_{\tau-1}(x_\tau)+\sigma_*^2}, \quad \hat{\lambda}_\tau = \sqrt{\hat{k}_{\tau-1}(x_\tau)+\hat{\sigma}^2},$$
$$\lambda = \sqrt{k^*_{\tau-1}(x^*)+\sigma_*^2}, \quad \hat{\lambda} = \sqrt{\hat{k}_{\tau-1}(x^*)+\hat{\sigma}^2}.$$

By Theorem 5 and the selection strategy of PI, with probability at least $1-\delta'$,

$$\begin{aligned}
\hat{f}^* - \mu^*_{\tau-1}(x_\tau) &< \hat{f}^* - \hat{\mu}_{\tau-1}(x_\tau) + a\lambda_\tau \\
&\leq \frac{\hat{f}^* - \hat{\mu}_{\tau-1}(x^*)}{\hat{\lambda}}\hat{\lambda}_\tau + a\lambda_\tau \\
&< \frac{\hat{f}^* - \mu^*_{\tau-1}(x^*) + a\lambda}{\hat{\lambda}}\hat{\lambda}_\tau + a\lambda_\tau \\
&< \frac{\hat{f}^* - \mu^*_{\tau-1}(x^*) + a\lambda}{\lambda}s\lambda_\tau + a\lambda_\tau \\
&= s\left(\frac{\hat{f}^* - \mu^*_{\tau-1}(x^*)}{\lambda} + a\right)\lambda_\tau + a\lambda_\tau,
\end{aligned}$$

where we set $a^2 = \frac{4\left(T+2\sqrt{(T-1)\log\frac{4}{\delta'}}+2\log\frac{4}{\delta'}-2/N\right)}{(N-T-1)\delta'}, b = \frac{1}{N-T}\log\frac{4}{\delta'}, s = \sqrt{\frac{1+2\sqrt{b}+2b}{1-2\sqrt{b}}}$.

Let $\eta = s(\frac{\hat{f}^* - \mu^*_{\tau-1}(x^*)}{\sqrt{k^*_{\tau-1}(x^*)+\sigma^2_*}} + a) + a + \zeta$ and $\delta' = \delta/3$. Then, with probability at least $1-\delta$, the simple regret of PI satisfy

$$r_T^{\text{PI}} < \eta\sqrt{\frac{2c\rho_T}{T\log(1+c\sigma_*^{-2})} + \sigma_*^2} + \zeta\sigma_*(1 - \frac{\sigma_*}{2\sqrt{c+\sigma_*^2}}).$$

□

Theorem 6 is a summary of Theorem 16 and Theorem 17. While the regret bounds provide us with some understanding of HyperBO in a specific setting, in practice, we need to query in a continuous input space that goes beyond the finite set of points present in the training dataset. It may or may not be possible to obtain data on a wide range of tasks to ensure $N > T+1$. In fact, in all of our experiment, this criterion on number of tasks is not satisfied. However, we still obtained good performance.

C.3 An Alternative Approach to Bound the Regret

In this section, we provide an alternative regret bound defined by the pre-trained GP in Theorem 18. This allows us to potentially examine the regret based on the observable parameters from the pre-trained GP. Here we set the GP-UCB acquisition function as

$$\alpha^{\text{UCB}}_{t-1}(x) = \hat{\mu}_{t-1}(x) + \beta\sqrt{\hat{k}_{t-1}(x)}, \tag{33}$$

which is more realistic than Eq. 31 in practical settings.

Theorem 18. *Assume $N > T + 4\log\frac{12}{\delta}$, $\delta \in (0,1)$, and there exists constant c such that $c \geq k^*(x)$ and $c \geq \hat{k}(x)$ for any $x \in \mathfrak{X}$. With probability at least $1-\delta$, GP-UCB in Eq. 33 obtains the simple regret*

$$r_T^{\textit{GP-UCB}} \leq 2\beta\sqrt{\frac{2c\rho_T}{T\log(1+c\hat{\sigma}^{-2})}} + \beta\hat{\sigma} + \zeta\sigma_*(1 - \frac{\sigma_*}{\sqrt{c+\sigma_*^2}}),$$

where $\beta = (a+\zeta)s$, $\zeta = \sqrt{2\log(\frac{3}{\delta})}$, $a^2 = \frac{12\left(T+2\sqrt{(T-1)\log\frac{12}{\delta}}+2\log\frac{12}{\delta}-2/N\right)}{(N-T-1)\delta}$, $b = \frac{1}{N-T}\log\frac{12}{\delta}$, $s = \sqrt{\frac{N}{(N-T)(1-2\sqrt{b})}}$, *and* $\rho_T = \max_{A\subseteq\mathfrak{X},|A|=T}\frac{1}{2}\log|\boldsymbol{I}+\hat{\sigma}^{-2}\hat{k}(A)|$.

Proof. Let $\tau = \arg\min_{t\in[T]}\hat{k}_{t-1}(x_t)$, $x^* = \arg\max_{x\in\mathfrak{X}} f(x)$ and $\hat{x} = x_{t'}$ where $t' = \arg\max_{t\in[T]} y_t$. We set $\zeta = \sqrt{2\log(\frac{1}{\delta'})}$ with $\delta' = \delta/3$. By Lemma 15 , with probability at least $1-\delta'$,

$$\begin{aligned} r_T^{\text{GP-UCB}} &= \left(f(x^*) - \max_{t\in[T]} f(x_t)\right) + \left(\max_{t\in[T]} f(x_t) - f(\hat{x})\right) \\ &\leq r + \zeta\sigma_*, \end{aligned}$$

where $r = f(x^*) - \max_{t\in[T]} f(x_t)$. We use the following short-hands for simplicity:

$$\begin{aligned} &\Delta = \sqrt{k^*_{\tau-1}(x^*)+\sigma_*^2} + \sqrt{k^*_{\tau-1}(x_\tau)+\sigma_*^2}, \quad \hat{\Delta} = \sqrt{\hat{k}_{\tau-1}(x^*)+\hat{\sigma}^2} + \sqrt{\hat{k}_{\tau-1}(x_\tau)+\hat{\sigma}^2}, \\ &\Delta_\mu = \mu^*_{\tau-1}(x^*) - \mu^*_{\tau-1}(x_\tau), \quad \hat{\Delta}_\mu = \hat{\mu}_{\tau-1}(x^*) - \hat{\mu}_{\tau-1}(x_\tau). \end{aligned}$$

Next we bound r. By Corollary 14, with probability at least $1-\delta'$,

$$\begin{aligned} r &\leq f(x^*) - f(x_\tau) \\ &\leq \Delta_\mu + \zeta\sqrt{k^*_{\tau-1}(x^*)} + \zeta\sqrt{k^*_{\tau-1}(x_\tau)}. \end{aligned}$$

By Theorem 5 and that $\tau \leq T$, the following inequality holds with probability at least $1-\delta'$,

$$\Delta_\mu < \hat{\Delta}_\mu + a\Delta \;\text{ and }\; \Delta < s\hat{\Delta},$$

where $a^2 = \frac{4\left(T+2\sqrt{(T-1)\log\frac{4}{\delta'}}+2\log\frac{4}{\delta'}-2/N\right)}{(N-T-1)\delta'}$, $b = \frac{1}{N-T}\log\frac{4}{\delta'}$, $s = \sqrt{\frac{N}{(N-T)(1-2\sqrt{b})}}$.

By Lemma 12 and that $c \geq \sup_{x\in\mathfrak{X}} k^*(x) \geq k^*_{\tau-1}(x^*)$, we have,

$$\sqrt{k^*_{\tau-1}(x^*)} + \sqrt{k^*_{\tau-1}(x_\tau)} \leq \Delta - \frac{\sigma_*^2}{\sqrt{c+\sigma_*^2}}$$

Notice that because of the acquisition strategy of GP-UCB, i.e., $\alpha^{\text{UCB}}_{\tau-1}(x) = \hat{\mu}_{\tau-1}(x) + \beta\sqrt{\hat{k}_{\tau-1}(x)}$, the following inequality holds,

$$\begin{aligned} \hat{\mu}_{\tau-1}(x^*) + \beta\sqrt{\hat{k}_{\tau-1}(x^*)} \leq \hat{\mu}_{\tau-1}(x_\tau) + \beta\sqrt{\hat{k}_{\tau-1}(x_\tau)}, \quad &\text{i.e.,} \\ \hat{\Delta}_\mu \leq \beta\left(\sqrt{\hat{k}_{\tau-1}(x_\tau)} - \sqrt{\hat{k}_{\tau-1}(x^*)}\right). \end{aligned}$$

Note that $\delta' = \delta/3$, $\beta = (a+\zeta)s$ and $\sqrt{\sigma_1^2+\sigma_2^2} < \sigma_1+\sigma_2$ for $\sigma_1, \sigma_2 > 0$. Hence, with probability at least $1-\delta$,

$$\begin{aligned}
r &< \Delta_\mu + \zeta\Delta - \frac{\zeta\sigma_*^2}{\sqrt{c+\sigma_*^2}} \\
&< \hat{\Delta}_\mu + (a+\zeta)s\hat{\Delta} - \frac{\zeta\sigma_*^2}{\sqrt{c+\sigma_*^2}} \\
&\leq \beta\left(\sqrt{\hat{k}_{\tau-1}(x_\tau)} - \sqrt{\hat{k}_{\tau-1}(x^*)}\right) + (a+\zeta)s\hat{\Delta} - \frac{\zeta\sigma_*^2}{\sqrt{c+\sigma_*^2}} \\
&< 2\beta(\sqrt{\hat{k}_{\tau-1}(x_\tau)}) + \beta\hat{\sigma} - \frac{\zeta\sigma_*^2}{\sqrt{c+\sigma_*^2}}.
\end{aligned}$$

By Corollary 11 and the fact that $\tau = \arg\min_{t\in[T]} \hat{k}_{t-1}(x_t)$, we have

$$\begin{aligned}
\hat{k}_{\tau-1}(x_\tau) &\leq \frac{1}{T}\sum_{t=1}^T \hat{k}_{t-1}(x_t) \\
&\leq \frac{1}{T}\sum_{t=1}^T \frac{c\log(1+\frac{c\hat{\sigma}^{-2}\hat{k}_{t-1}(x_t)}{c})}{\log(1+c\hat{\sigma}^{-2})} \\
&= \frac{c}{T\log(1+c\hat{\sigma}^{-2})}\sum_{t=1}^T \log(1+\hat{\sigma}^{-2}\hat{k}_{t-1}(x_t)).
\end{aligned}$$

Notice that here Corollary 11 applies because $0 \leq \hat{k}_{\tau-1}(x_\tau) \leq c$.

By Lemma 13, if we create an imaginary function $f' \sim \mathcal{GP}(\hat{\mu}, \hat{k})$ with observation noise $\hat{\sigma}^2$, we get $I(f'(\boldsymbol{x}_T); \boldsymbol{y}_T) = \frac{1}{2}\sum_{t=1}^T \log(1+\hat{\sigma}^{-2}\hat{k}_{t-1}(x_t)) \leq \rho_T$, so

$$\hat{k}_{\tau-1}(x_\tau) \leq \frac{2c\rho_T}{T\log(1+c\hat{\sigma}^{-2})},$$

which implies

$$r_T^{\text{GP-UCB}} < 2\beta\sqrt{\frac{2c\rho_T}{T\log(1+c\hat{\sigma}^{-2})}} + \beta\hat{\sigma} + \zeta\sigma_*(1-\frac{\sigma_*}{\sqrt{c+\sigma_*^2}}).$$

Note that ρ_T is the max information gain over T observations of the pre-trained GP. The last term, $\zeta\sigma_*(1-\frac{\sigma_*}{\sqrt{c+\sigma_*^2}})$, involves the ground truth noise parameter, and it is irreducible due to the assumed observation noise. □

Appendix D. Experiment Details and More Results

In this section, we provide more empirical results on the impact of objective functions and acquisition functions in HyperBO. The experiment setups on PD1 are the same as §7.4.1: offline and holding out related tasks. The experiment setups on HPO-B are the same as §7.6, which follows the original settings from Pineda-Arango et al. (2021).

D.1 More Results on the PD1 Online Tuning Tasks

Figure 22 is an extension of Figure 14 and it includes more methods that are evaluated on the PD1 online tuning tasks. Due to practical constraints, we set HyperBO variants and STBO to share exactly the same GP-UCB acquisition function as STBOH, MIMO and RFGP. We used 1.8 as the UCB coefficient for all methods.

Overall, HyperBO methods achieved more robust and better results than baselines on most of the tasks. It was very difficult for STBO, MIMO and RFGP to recover from points with infeasible evaluations, since they kept "exploiting" inputs with infeasible evaluations and as a result performed poorly. MIMO, in particular, got stuck in all of the online experiments. On ImageNet ResNet50 512, LM1B Transfoermer 2048, Uniref50 Transformer 128 and WMT XFormer 64, all of the hyperparameters selected by MIMO resulted in infeasible training runs, and so its regret curves did not exist for these tasks.

D.2 Impact of Acquisition Functions

We tested 3 acquisition functions in our experiments: PI from §7.2, EI (Šaltenis, 1971; Močkus, 1974) and GP-UCB with coefficient $\beta^{\frac{1}{2}} = 3$ in Eq. (6) of Srinivas et al. (2010). We conducted these experiments on PD1 and HPO-B for all available HyperBO variants. The results for PD1 are shown in Figure 23 and HPO-B in Figure 24. Both sets of results show that HyperBO methods are not very sensitive to these 3 choices of acquisition functions, and overall PI has slightly better and more stable performance than EI and GP-UCB.

D.3 Impact of Mean and Kernel Structures

One open question we did not dive into in §5 was how to determine the functional structures for mean and kernel functions to be used for pre-training. Here we tested different mean and kernel structures to evaluate their impacts on pre-trained GPs and BO results.

In Figure 25 on PD1, we show the comparisons between 9 different mean and kernel combinations. The choices for the mean function include zero mean, constant mean and a linear MLP. We used Matérn52 kernel on top of the same MLP as the linear MLP mean function. The MLP architectures include 1-hidden layer, either 4 or 8 neurons, or 2-hidden layer, with 32 neurons per layer. Note that the input dimension of PD1 is 4. We abbreviate these structures to "Zero (4,)" denoting zero mean with (4,) architecture for the Matérn52 MLP kernel, "Linear MLP (32, 32)" denoting linear MLP mean function with (32, 32) architecture for both the mean and the Matérn52 MLP kernel, etc.

For H-NLL, Zero (4,) is the simplest model among all and it performed worse than other structures. Across BO iterations, Linear MLP (8,) and Linear MLP (32, 32) achieved better performance than HyperBO with other mean / kernel architectures. Towards 100 BO iterations, Linear MLP (8,), Linear MLP (32, 32), Zero (32, 32) and Constant (8,) achieved roughly the same performance.

For H-EKL, the performance gaps for these different structures are larger than H-NLL. Using zero mean with any of the kernel structures resulted in much worse overall BO performance than other structures. For other structures, in general, HyperBO using a linear MLP mean function performed better than its constant mean counterparts. Linear MLP (32, 32) and Constant (32, 32) achieved better regrets towards finishing 100 BO iterations.

In Figure 26 on HPO-B, similar to the experiment on PD1, we show the comparisons for several different mean and kernel structures, also with Matérn52 kernel with MLP features. With gradually

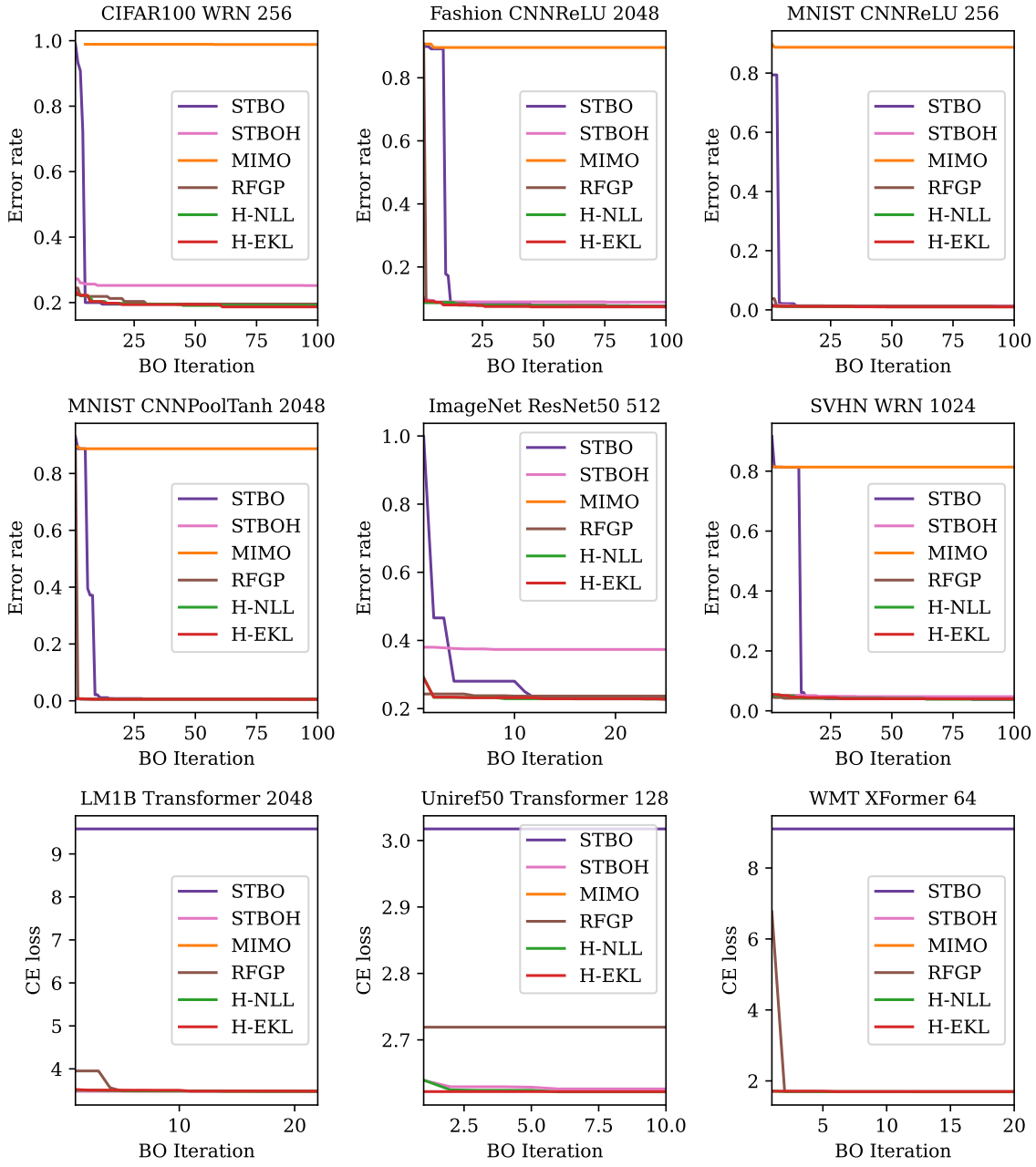

Figure 22: Extension of Figure 14 to include more methods. STBO, MIMO, RFGP encountered many infeasible evaluations due to their acquisition strategies.

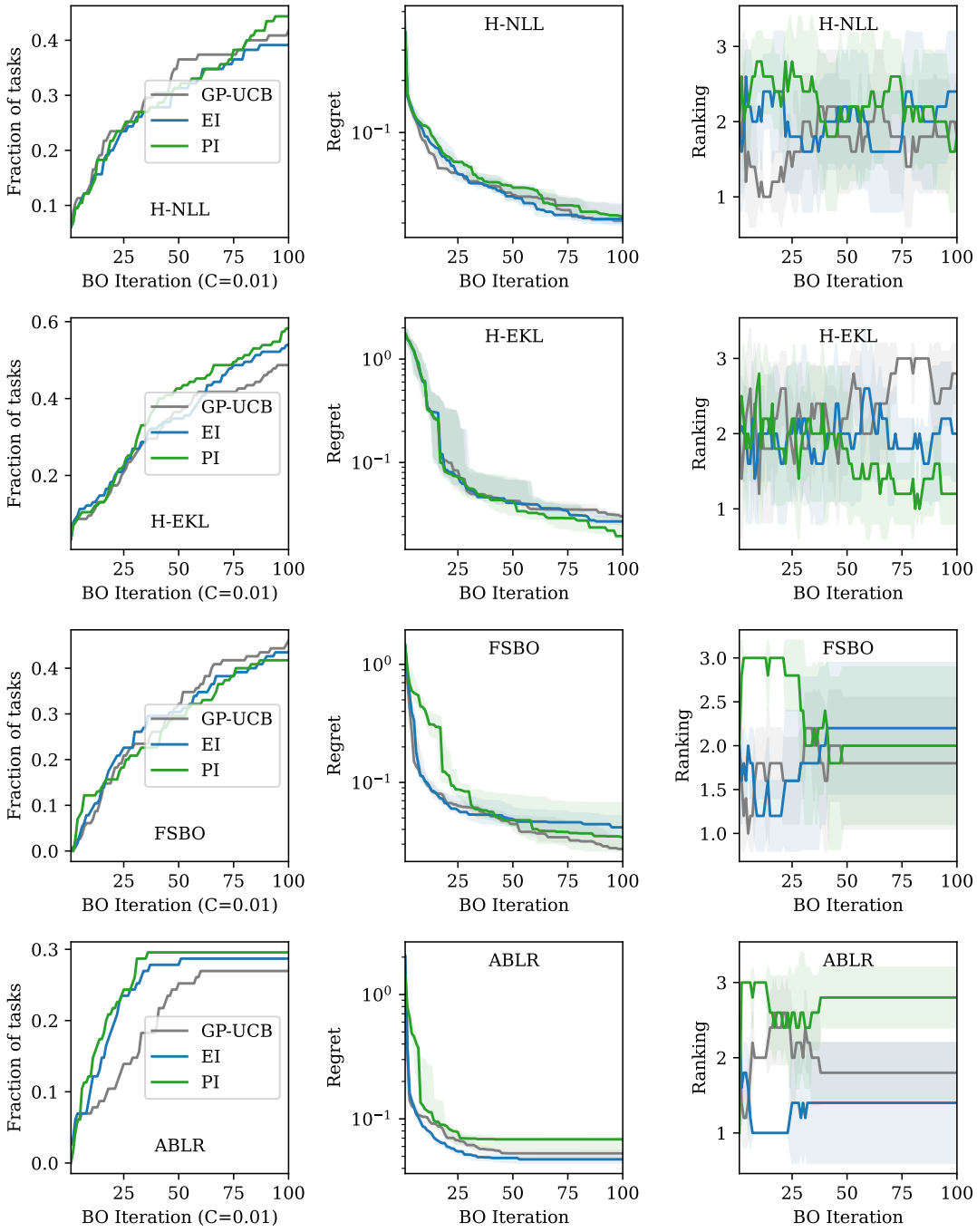

Figure 23: PD1. We compare the performance of 3 different acquisition functions under 4 variants of HyperBO. The results of these acquisition functions are very similar and within each other's confidence intervals. The performance profiles (first column) show that PI (with threshold 0.1) has slightly better and more stable performance than EI and GP-UCB.

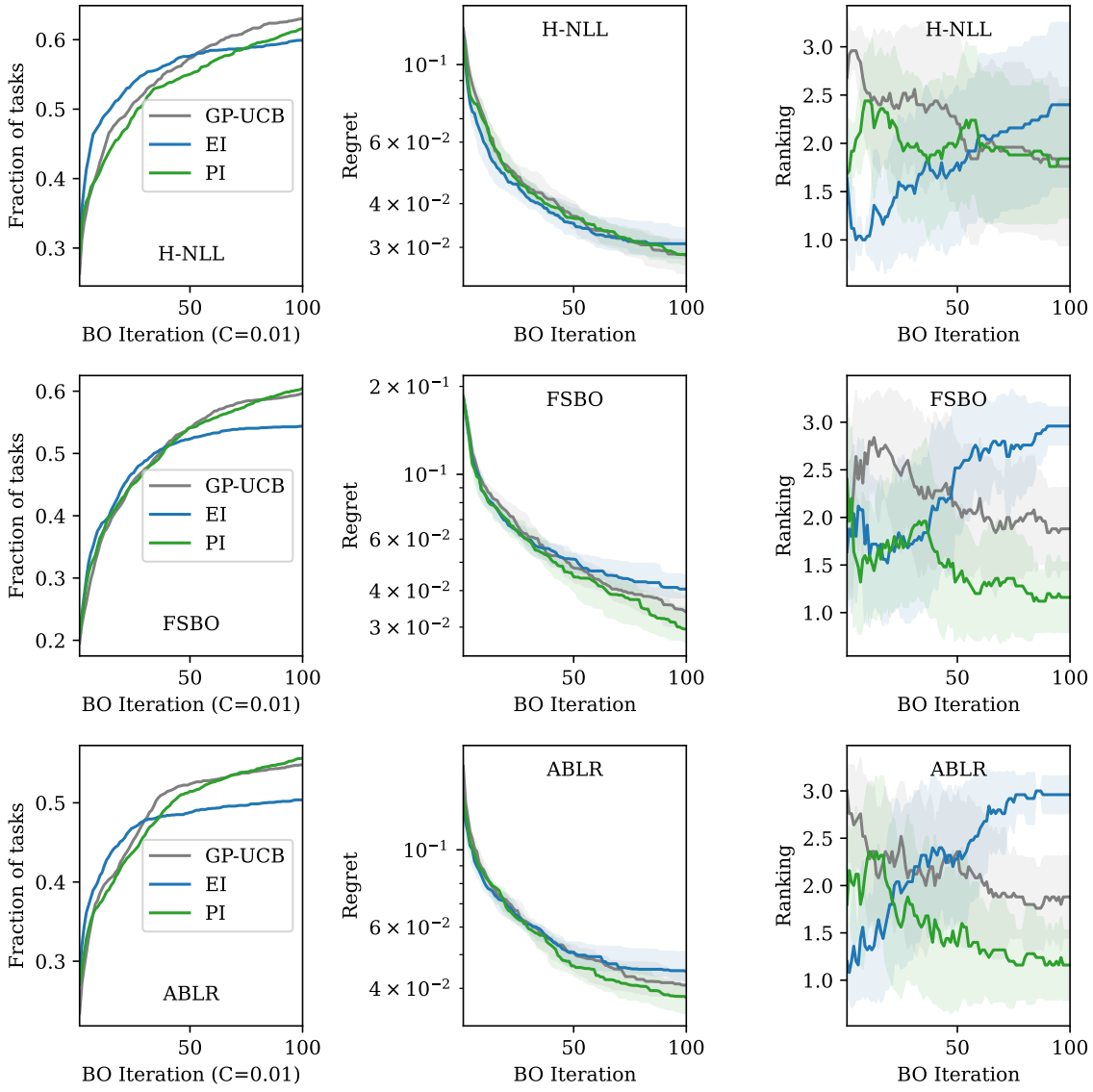

Figure 24: HPO-B. We compare the performance of 3 different acquisition functions under 3 variants of HyperBO. There are no significant differences. PI (with threshold 0.1) and GP-UCB (with coefficient 3) achieved slightly better results than EI.

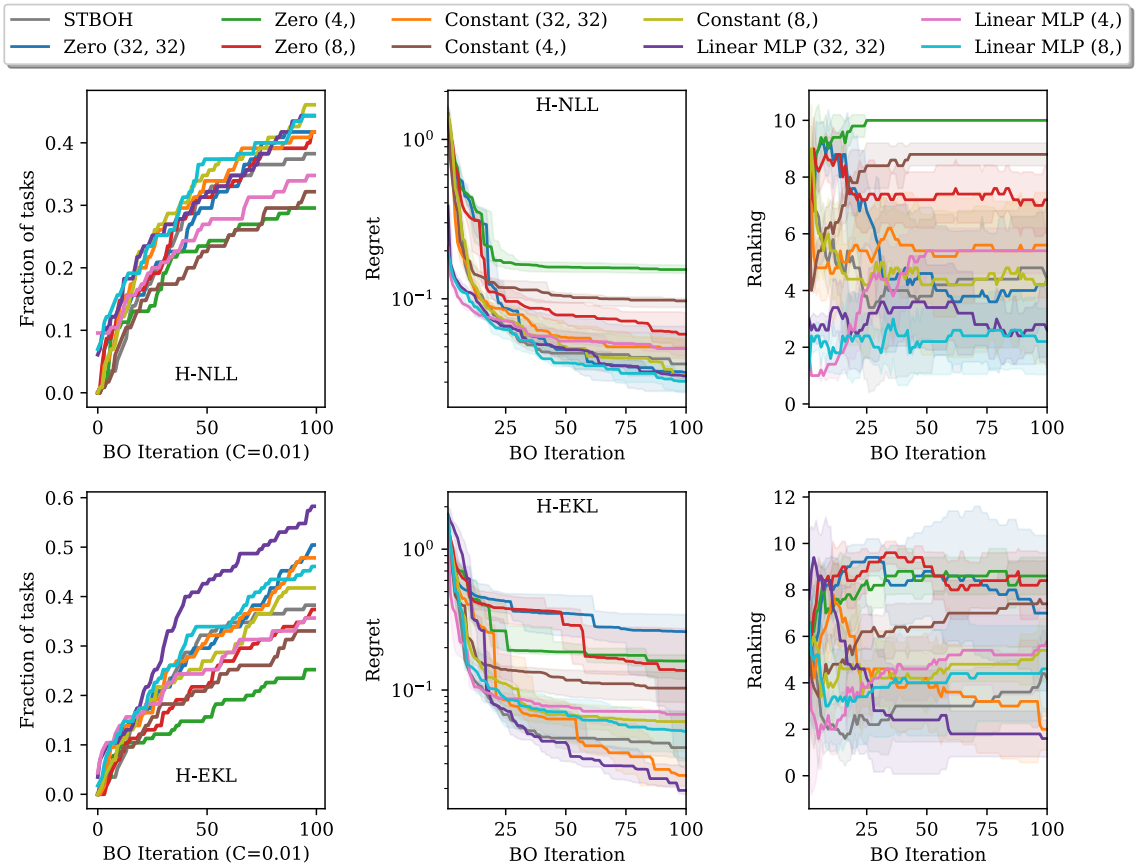

Figure 25: PD1. We compare the performance of 10 different mean and kernel structures of HyperBO. The kernel is Matérn52 with MLP features. The legend means mean function type and the numbers mean the feature size. Acquisition function is PI.

more complex structures, in general, H-NLL was able to obtain gradually better regrets. Similar to the results in PD1, HyperBO with zero mean performed relatively worse. Across BO iterations, Linear MLP (128, 128) and Linear MLP (32, 32) achieved more stable and better results.

Overall, Figure 25 and Figure 26 show that HyperBO using a linear MLP mean function often outperforms using zero mean. While there exist performance differences among different mean and kernel structures, with relatively complex architectures, the regrets did not change much in our experiments.

D.4 More Results on HPO-B

For readers who are interested in studying the details of the HPO-B experiment, we present the results separately for each of the 16 search spaces of HPO-B. The experiment setups are the same as §7.6.1. Figure 27 shows the performance profiles, Figure 28 shows the regret plots, and Figure 29 shows the ranking plots.

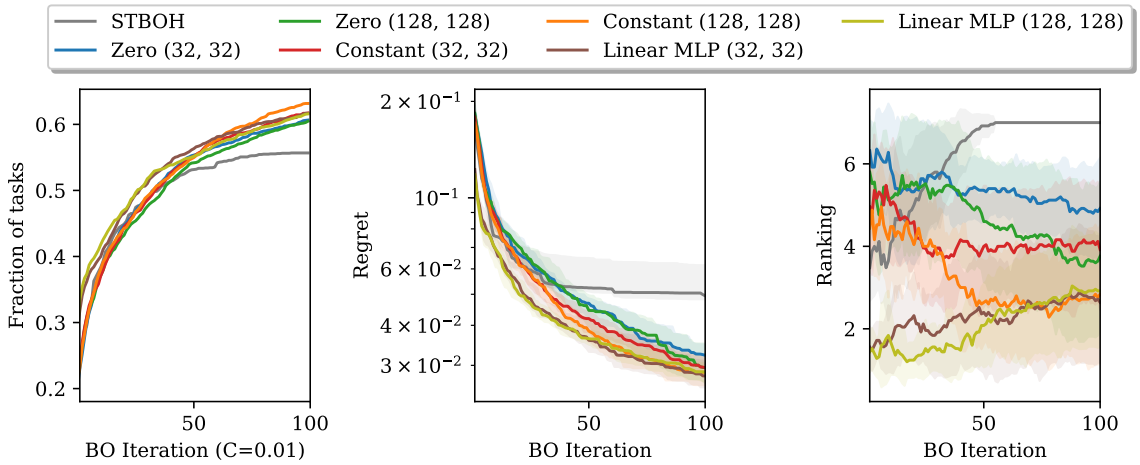

Figure 26: HPOB with H-NLL. We compare the performance of 7 different mean and kernel structures of HyperBO. The kernel is Matérn52 with MLP features. The legend denotes mean function type and the numbers are the feature sizes. Acquisition function is PI.

D.5 MAF Implementation Details

We compared to (Volpp et al., 2020) using the code and default hyperparameters provided by the authors.[9] This method assumes the availability of the optimal set of GP hyperparameters for each task (including the task used for evaluation). Following Volpp et al. (2020), these GP hyperparameters for the MAF algorithm are learned by optimizing the marginal likelihood on each training and evaluation task using the GPy library. Given that MAF takes significantly longer to run than HyperBO and other baselines, sub-dataset in each task was evaluated using limited random seeds.

Each neural acquisition function was trained for a total of 1000 iterations. As was done in (Volpp et al., 2020), we selected the optimal training iteration for the neural acquisition function by cross-validation on the transfer learning tasks; in this case, we randomly sampled 3 transfer learning task, and chose the training iteration with the lowest average simple regret.

Finally, to make use of the MAF code, we also had to ensure that (a) each task had the same number of evaluation points, and (b) that there were no duplicated tuning parameters. For this reason, we first removed all duplicate hyperparameters within each sub-dataset, then capped each sub-dataset to the first 1559 points (the size of the smallest sub-dataset) while retaining the best possible datapoint.

9. https://github.com/boschresearch/MetaBO

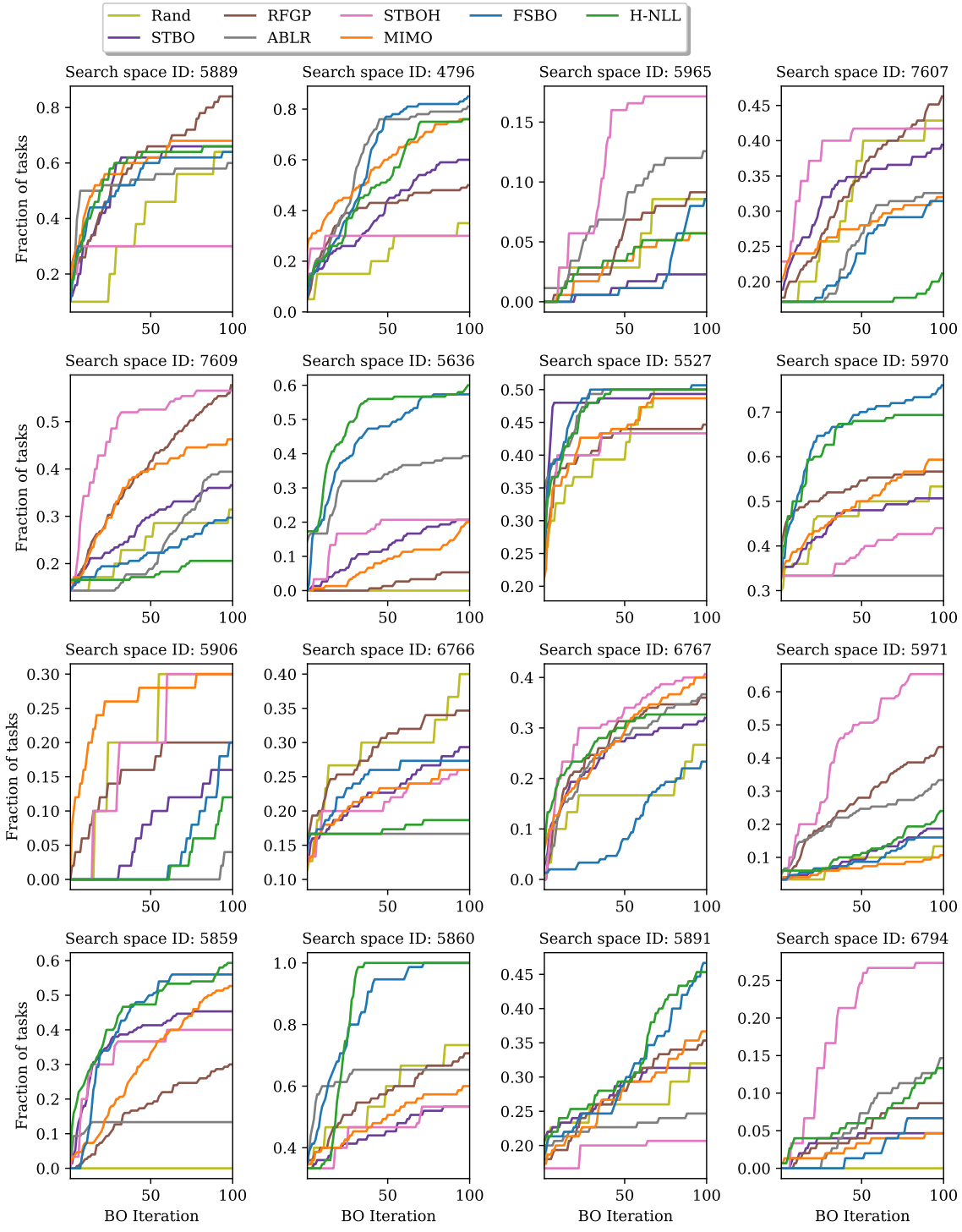

Figure 27: The performance profiles for each of the 16 search spaces in HPO-B. We use $C = 0.001$ as the threshold.

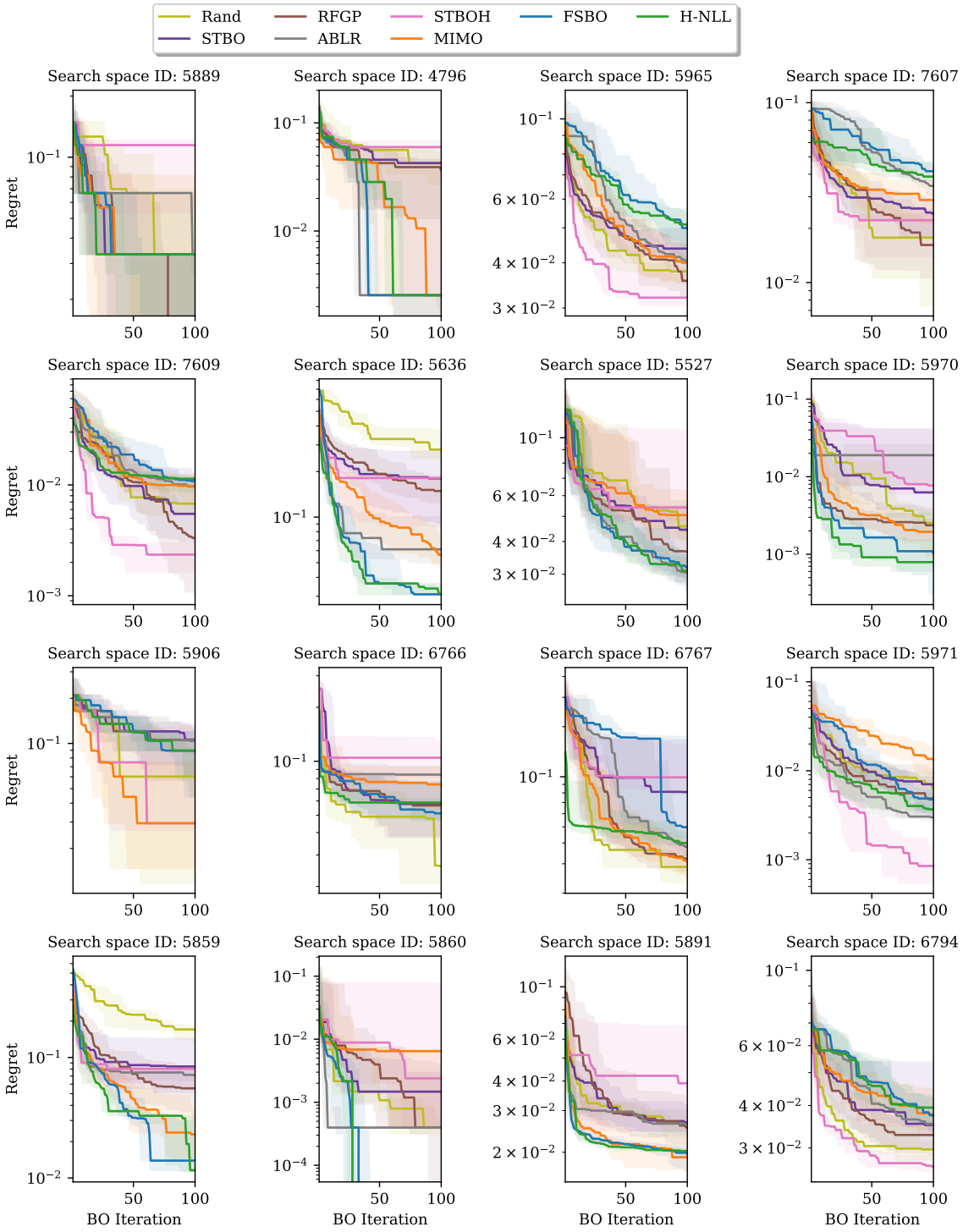

Figure 28: The regrets for each of the 16 search spaces in HPO-B.

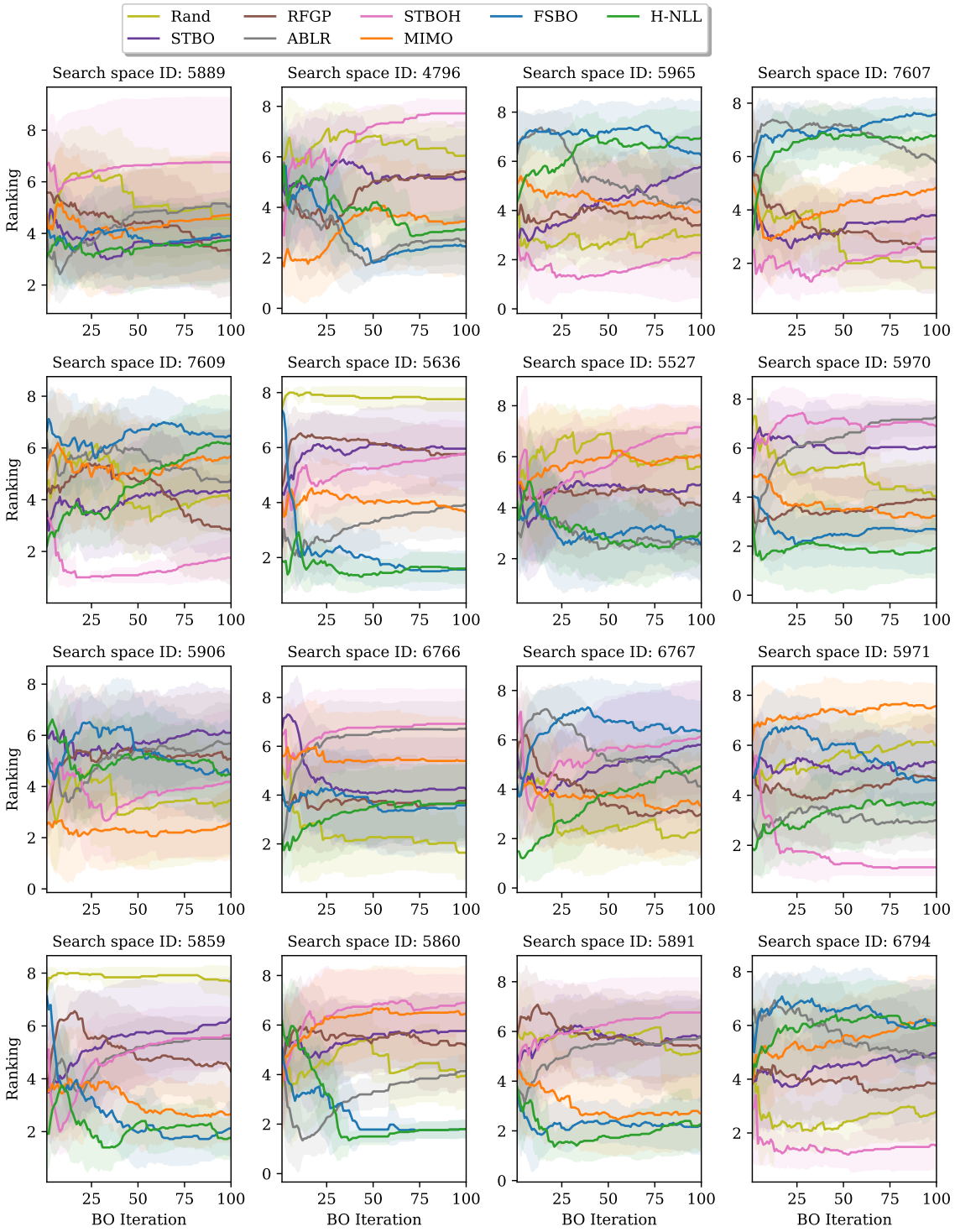

Figure 29: The ranking plots for each of the 16 search spaces in HPO-B.

References

Theodore Wilbur Anderson. *An Introduction to Multivariate Statistical Analysis*. Wiley New York, 1958.

Peter Auer. Using confidence bounds for exploitation-exploration tradeoffs. *Journal of Machine Learning Research (JMLR)*, 3:397–422, 2002.

Igor Babuschkin, Kate Baumli, Alison Bell, Surya Bhupatiraju, Jake Bruce, Peter Buchlovsky, David Budden, Trevor Cai, Aidan Clark, Ivo Danihelka, Claudio Fantacci, Jonathan Godwin, Chris Jones, Ross Hemsley, Tom Hennigan, Matteo Hessel, Shaobo Hou, Steven Kapturowski, Thomas Keck, Iurii Kemaev, Michael King, Markus Kunesch, Lena Martens, Hamza Merzic, Vladimir Mikulik, Tamara Norman, John Quan, George Papamakarios, Roman Ring, Francisco Ruiz, Alvaro Sanchez, Rosalia Schneider, Eren Sezener, Stephen Spencer, Srivatsan Srinivasan, Luyu Wang, Wojciech Stokowiec, and Fabio Viola. The DeepMind JAX Ecosystem, 2020. URL `http://github.com/deepmind`.

François Bachoc. Asymptotic analysis of maximum likelihood estimation of covariance parameters for Gaussian processes: an introduction with proofs. In *Advances in Contemporary Statistics and Econometrics: Festschrift in Honor of Christine Thomas-Agnan*, pages 283–303. Springer, 2021.

Rémi Bardenet, Mátyás Brendel, Balázs Kégl, and Michele Sebag. Collaborative hyperparameter tuning. In *International Conference on Machine Learning (ICML)*, 2013.

J Baxter. A Bayesian/information theoretic model of bias learning. In *Conference on Learning Theory (COLT)*, 1996.

Felix Berkenkamp, Angela P Schoellig, and Andreas Krause. No-regret Bayesian optimization with unknown hyperparameters. *Journal of Machine Learning Research (JMLR)*, 20(50):1–24, 2019.

Ilija Bogunovic and Andreas Krause. Misspecified Gaussian process bandit optimization. *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Edwin V Bonilla, Kian Chai, and Christopher Williams. Multi-task Gaussian process prediction. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2007.

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

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL `http://github.com/google/jax`.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

David R Burt, Sebastian W Ober, Adrià Garriga-Alonso, and Mark van der Wilk. Understanding variational inference in function-space. In *3rd Symposium on Advances in Approximate Bayesian Inference*, 2020.

Rich Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

Yutian Chen, Xingyou Song, Chansoo Lee, Zi Wang, Qiuyi Zhang, David Dohan, Kazuya Kawakami, Greg Kochanski, Arnaud Doucet, Marc'Aurelio Ranzato, Sagi Perel, and Nando de Freitas. Towards learning universal hyperparameter optimizers with transformers. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Dami Choi, Christopher J Shallue, Zachary Nado, Jaehoon Lee, Chris J Maddison, and George E Dahl. On empirical comparisons of optimizers for deep learning. *arXiv:1910.05446 [cs.LG]*, 2019.

Emile Contal, Vianney Perchet, and Nicolas Vayatis. Gaussian process optimization with mutual information. In *International Conference on Machine Learning (ICML)*, 2014.

Thomas M Cover and Joy A Thomas. *Elements of Information Theory*. John Wiley & Sons, 2006.

Alexander I Cowen-Rivers, Wenlong Lyu, Rasul Tutunov, Zhi Wang, Antoine Grosnit, Ryan Rhys Griffiths, Alexandre Max Maraval, Hao Jianye, Jun Wang, Jan Peters, and Haitham Bou-Ammar. HEBO: Pushing the limits of sample-efficient hyperparameter optimisation. *Journal or Artificial Intelligence Research (JAIR)*, 74:1269–1349, 2022.

Walter Daelemans, Antal Van den Bosch, et al. *Memory-based Language Processing*. Cambridge University Press, 2005.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2019.

Joshua V Dillon, Ian Langmore, Dustin Tran, Eugene Brevdo, Srinivas Vasudevan, Dave Moore, Brian Patton, Alex Alemi, Matt Hoffman, and Rif A Saurous. Tensorflow distributions. *arXiv:1711.10604 [cs.LG]*, 2017.

Elizabeth D Dolan and Jorge J Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91:201–213, 2002.

Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. DeCAF: A deep convolutional activation feature for generic visual recognition. In *International Conference on Machine Learning (ICML)*, 2014.

Danny Drieß, Peter Englert, and Marc Toussaint. Constrained Bayesian optimization of combined interaction force/task space controllers for manipulations. In *International Conference on Robotics and Automation (ICRA)*, 2017.

Morris L. Eaton. *Multivariate statistics: A vector space approach*. Institute of Mathematical Statistics, Beachwood, Ohio, USA, 2007.

Zhou Fan, Xinran Han, and Zi Wang. HyperBO+: Pre-training a universal hierarchical Gaussian process prior for Bayesian optimization. *NeurIPS Workshop on Gaussian Processes, Spatiotemporal Modeling, and Decision-making Systems*, 2022.

Zhou Fan, Xinran Han, and Zi Wang. Transfer learning for bayesian optimization on heterogeneous search spaces. *Transactions on Machine Learning Research (TMLR)*, 2024.

Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2015.

Matthias Feurer, Benjamin Letham, Frank Hutter, and Eytan Bakshy. Practical transfer learning for Bayesian optimization. *arXiv:1802.02219 [stat.ML]*, 2018.

Jacob Gardner, Matt Kusner, Zhixiang, Kilian Weinberger, and John Cunningham. Bayesian optimization with inequality constraints. In *International Conference on Machine Learning (ICML)*, 2014.

Jacob Gardner, Chuan Guo, Kilian Weinberger, Roman Garnett, and Roger Grosse. Discovering and exploiting additive structure for Bayesian optimization. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2017.

Roman Garnett. *Bayesian optimization*. Cambridge University Press, 2023.

Michael Gelbart. *Constrained Bayesian Optimization and Applications*. PhD thesis, Harvard University, 2015.

Michael A. Gelbart, Jasper Snoek, and Ryan P. Adams. Bayesian optimization with unknown constraints. In *Uncertainty in Artificial Intelligence*, 2014.

Justin M. Gilmer, George E. Dahl, and Zachary Nado. init2winit: a JAX codebase for initialization, optimization, and tuning research, 2021. URL `http://github.com/google/init2winit`.

Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Elliot Karro, and D. Sculley. Google Vizier: A service for black-box optimization. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2017.

Ryan-Rhys Griffiths and José Miguel Hernández-Lobato. Constrained Bayesian optimization for automatic chemical design using variational autoencoders. *Chemical Science*, 11(2):577–586, 2020.

Thomas Griffiths, Chris Lucas, Joseph Williams, and Michael Kalish. Modeling human function learning with gaussian processes. *Advances in Neural Information Processing Systems (NeurIPS)*, 2008.

Mengyang Gu, Xiaojing Wang, and James O Berger. Robust Gaussian stochastic process emulation. *The Annals of Statistics*, 46(6A):3038–3066, 2018.

Nikolaus Hansen, Anne Auger, Raymond Ros, Olaf Mersmann, Tea Tušar, and Dimo Brockhoff. COCO: A platform for comparing continuous optimizers in a black-box setting. *Optimization Methods and Software*, 36(1):114–144, 2021.

Marton Havasi, Rodolphe Jenatton, Stanislav Fort, Jeremiah Zhe Liu, Jasper Snoek, Balaji Lakshminarayanan, Andrew M Dai, and Dustin Tran. Training independent subnetworks for robust prediction. In *International Conference on Learning Representations (ICLR)*, 2021.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

Jonathan Heek, Anselm Levskaya, Avital Oliver, Marvin Ritter, Bertrand Rondepierre, Andreas Steiner, and Marc van Zee. Flax: A neural network library and ecosystem for JAX, 2020. URL `http://github.com/google/flax`.

Philipp Hennig and Christian J Schuler. Entropy search for information-efficient global optimization. *Journal of Machine Learning Research (JMLR)*, 13:1809–1837, 2012.

José Miguel Hernández-Lobato, Matthew W Hoffman, and Zoubin Ghahramani. Predictive entropy search for efficient global optimization of black-box functions. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2014.

José Miguel Hernández-Lobato, Michael A. Gelbart, Ryan P. Adams, Matthew W. Hoffman, and Zoubin Ghahramani. A general framework for constrained Bayesian optimization using information-based search. *Journal of Machine Learning Research (JMLR)*, 17(160):1–53, 2016.

E. T. Jaynes. *Probability theory: The logic of science*. Cambridge University Press, 2003.

Norman P Jouppi, Doe Hyun Yoon, Matthew Ashcraft, Mark Gottscho, Thomas B Jablin, George Kurian, James Laudon, Sheng Li, Peter Ma, Xiaoyu Ma, et al. Ten lessons from three generations shaped Google's TPUv4i. In *ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021.

Tarun Kathuria, Amit Deshpande, and Pushmeet Kohli. Batched Gaussian process bandit optimization via determinantal point processes. *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

Charles Kemp and Joshua B Tenenbaum. The discovery of structural form. *Proceedings of the National Academy of Sciences*, 105(31):10687–10692, 2008.

Beomjoon Kim, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Learning to guide task and motion planning using score-space representation. In *International Conference on Robotics and Automation (ICRA)*, 2017.

Beomjoon Kim, Zi Wang, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Learning to guide task and motion planning using score-space representation. *International Journal of Robotics Research (IJRR)*, 38(7):793–812, 2019.

Samuel Kim, Peter Y Lu, Charlotte Loh, Jamie Smith, Jasper Snoek, and Marin Soljačić. Deep learning for Bayesian optimization of scientific problems with high-dimensional structure. *Transactions on Machine Learning Research (TMLR)*, 2022.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

Lars Kotthoff, Chris Thornton, Holger H. Hoos, Frank Hutter, and Kevin Leyton-Brown. Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA. *Journal of Machine Learning Research (JMLR)*, 18(25):1–5, 2017.

Andreas Krause and Cheng S Ong. Contextual Gaussian process bandit optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2011.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

Harold J Kushner. A versatile stochastic model of a function of unknown and time varying form. *Journal of Mathematical Analysis and Applications*, 5(1):150–167, 1962.

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

Balaji Lakshminarayanan, Daniel M Roy, and Yee Whye Teh. Mondrian forests for large-scale regression when uncertainty matters. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2016.

Rémi Lam, Matthias Poloczek, Peter Frazier, and Karen E Willcox. Advances in Bayesian optimization with applications in aerospace engineering. In *AIAA Non-Deterministic Approaches Conference*, page 1656, 2018.

Beatrice Laurent and Pascal Massart. Adaptive estimation of a quadratic functional by model selection. *Annals of Statistics*, pages 1302–1338, 2000.

Yann LeCun, Corinna Cortes, and CJ Burges. MNIST handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research (JMLR)*, 18(1):6765–6816, 2017.

Dong C Liu and Jorge Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45(1):503–528, 1989.

Jeremiah Zhe Liu, Zi Lin, Shreyas Padhy, Dustin Tran, Tania Bedrax-Weiss, and Balaji Lakshminarayanan. Simple and principled uncertainty estimation with deterministic deep learning via distance awareness. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Gustavo Malkomes and Roman Garnett. Automating Bayesian optimization with Bayesian optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

Gustavo Malkomes, Charles Schaff, and Roman Garnett. Bayesian optimization for automated model selection. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

Thomas P. Minka and Rosalind W. Picard. Learning how to learn is learning with point sets. Technical report, MIT Media Lab, 1997.

J. Močkus. On Bayesian methods for seeking the extremum. In *Optimization Techniques IFIP Technical Conference*, 1974.

Yurii E. Nesterov. A method of solving a convex programming problem with convergence rate $\mathrm{O}\big(1/k^2\big)$. In *Doklady Akademii Nauk*, volume 269, pages 543–547. Russian Academy of Sciences, 1983.

ChangYong Oh, Efstratios Gavves, and Max Welling. Bock: Bayesian optimization with cylindrical kernels. In *International Conference on Machine Learning (ICML)*, 2018.

Mahamed GH Omran, Andries P Engelbrecht, and Ayed Salman. An overview of clustering methods. *Intelligent Data Analysis*, 11(6):583–605, 2007.

Michael Arthur Leopold Pearce, Matthias Poloczek, and Juergen Branke. Bayesian optimization allowing for common random numbers. *Operations Research*, 70(6):3457–3472, 2022.

Valerio Perrone, Rodolphe Jenatton, Matthias Seeger, and Cédric Archambeau. Scalable hyperparameter transfer learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

Sebastian Pineda-Arango, Hadi S. Jomaa, Martin Wistuba, and Josif Grabocka. 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.

Matthias Poloczek, Jialei Wang, and Peter I Frazier. Warm starting Bayesian optimization. In *Winter Simulation Conference (WSC)*, 2016.

Matthias Poloczek, Jialei Wang, and Peter Frazier. Multi-information source optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Edward O Pyzer-Knapp. Bayesian optimization for accelerated drug discovery. *IBM Journal of Research and Development*, 62(6):2–1, 2018.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research (JMLR)*, 2020.

Ali Rahimi, Benjamin Recht, et al. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2007.

Carl Edward Rasmussen and Christopher K.I. Williams. *Gaussian processes for machine learning*. The MIT Press, 2006.

Lassi Roininen, Mark Girolami, Sari Lasanen, and Markku Markkanen. Hyperpriors for Matérn fields with applications in Bayesian inversion. *Inverse Problems and Imaging*, 13(1):1–29, 2019.

Paul Rolland, Jonathan Scarlett, Ilija Bogunovic, and Volkan Cevher. High-dimensional Bayesian optimization via additive models with overlapping groups. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2018.

Jonas Rothfuss, Vincent Fortuin, Martin Josifoski, and Andreas Krause. PACOH: Bayes-optimal meta-learning with PAC-guarantees. In *International Conference on Machine Learning (ICML)*, 2021.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision (IJCV)*, 115(3): 211–252, 2015.

Stuart J. Russell and Peter Norvig. *Artificial intelligence: A modern approach*. Prentice Hall, 2003.

Shiori Sagawa, Pang Wei Koh, Tatsunori B Hashimoto, and Percy Liang. Distributionally robust neural networks for group shifts: On the importance of regularization for worst-case generalization. In *International Conference on Learning Representations (ICLR)*, 2020.

David Salinas, Huibin Shen, and Valerio Perrone. A quantile-based approach for hyperparameter transfer learning. In *International Conference on Machine Learning (ICML)*, 2020.

V Šaltenis. On a method of multi-extremal optimization. *Automatics and Computers (Avtomatika i Vychislitelnayya Tekchnika)*, 3:33–38, 1971.

Cullen Schaffer. A conservation law for generalization performance. In *International Conference on Machine Learning (ICML)*, 1994.

J Schmidhuber. On learning how to learn learning strategies. Technical report, FKI-198-94 (revised), 1995.

Eric Schulz, Maarten Speekenbrink, José Miguel Hernández-Lobato, Zoubin Ghahramani, and Samuel J Gershman. Quantifying mismatch in Bayesian optimization. In *NeurIPS Workshop on Bayesian Optimization: Black-box Optimization and Beyond*, 2016.

Benjamin J Shields, Jason Stevens, Jun Li, Marvin Parasram, Farhan Damani, Jesus I Martinez Alvarado, Jacob M Janey, Ryan P Adams, and Abigail G Doyle. Bayesian reaction optimization as a tool for chemical synthesis. *Nature*, 590(7844):89–96, 2021.

Jasper Snoek. *Bayesian Optimization and Semiparametric Models with Applications to Assistive Technology*. PhD thesis, University of Toronto, 2013.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2012.

Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Mostofa Patwary, Mr Prabhat, and Ryan Adams. Scalable Bayesian optimization using deep neural networks. In *International Conference on Machine Learning (ICML)*, 2015.

Jost Tobias Springenberg, Aaron Klein, Stefan Falkner, and Frank Hutter. Bayesian optimization with robust Bayesian neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

Niranjan Srinivas, Andreas Krause, Sham M Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. In *International Conference on Machine Learning (ICML)*, 2010.

Shengyang Sun, Guodong Zhang, Jiaxin Shi, and Roger Grosse. Functional variational Bayesian neural networks. In *International Conference on Learning Representations (ICLR)*, 2019.

Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International Conference on Machine Learning (ICML)*, 2013.

Kevin Swersky, Jasper Snoek, and Ryan P Adams. Multi-task Bayesian optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2013.

Alexander Terenin, David R. Burt, Artem Artemev, Seth Flaxman, Mark van der Wilk, Carl Edward Rasmussen, and Hong Ge. Numerically stable sparse Gaussian processes via minimum separation using cover trees. *Journal of Machine Learning Research (JMLR)*, 25(26):1–36, 2024.

Petru Tighineanu, Kathrin Skubch, Paul Baireuther, Attila Reiss, Felix Berkenkamp, and Julia Vinogradska. Transfer learning with gaussian processes for bayesian optimization. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2022.

Michalis Titsias. Variational learning of inducing variables in sparse Gaussian processes. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2009.

Ba-Hien Tran, Simone Rossi, Dimitrios Milios, and Maurizio Filippone. All you need is a good functional prior for Bayesian deep learning. *Journal of Machine Learning Research (JMLR)*, 23 (1):3210–3265, 2022.

Ryan Turner, David Eriksson, Michael McCourt, Juha Kiili, Eero Laaksonen, Zhen Xu, and Isabelle Guyon. Bayesian optimization is superior to random search for machine learning hyperparameter tuning: Analysis of the black-box optimization challenge 2020. In *NeurIPS 2020 Competition and Demonstration Track*, Proceedings of Machine Learning Research, 2021.

Mark Van der Wilk, Carl Edward Rasmussen, and James Hensman. Convolutional Gaussian processes. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

S Vichy N Vishwanathan, Nicol N Schraudolph, Risi Kondor, and Karsten M Borgwardt. Graph kernels. *Journal of Machine Learning Research (JMLR)*, 11:1201–1242, 2010.

Michael Volpp, Lukas P Fröhlich, Kirsten Fischer, Andreas Doerr, Stefan Falkner, Frank Hutter, and Christian Daniel. Meta-learning acquisition functions for transfer learning in Bayesian optimization. In *International Conference on Learning Representations (ICLR)*, 2020.

Zi Wang and Stefanie Jegelka. Max-value entropy search for efficient Bayesian optimization. In *International Conference on Machine Learning (ICML)*, 2017.

Zi Wang, Bolei Zhou, and Stefanie Jegelka. Optimization as estimation with Gaussian processes in bandit settings. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2016a.

Zi Wang, Stefanie Jegelka, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Focused model-learning and planning for non-Gaussian continuous state-action systems. In *International Conference on Robotics and Automation (ICRA)*, 2017a.

Zi Wang, Chengtao Li, Stefanie Jegelka, and Pushmeet Kohli. Batched high-dimensional Bayesian optimization via structural kernel learning. In *International Conference on Machine Learning (ICML)*, 2017b.

Zi Wang, Clement Gehring, Pushmeet Kohli, and Stefanie Jegelka. Batched large-scale Bayesian optimization in high-dimensional spaces. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2018a.

Zi Wang, Beomjoon Kim, and Leslie Pack Kaelbling. Regret bounds for meta Bayesian optimization with an unknown Gaussian process prior. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018b.

Zi Wang, Caelan Reed Garrett, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Learning compositional models of robot skills for task and motion planning. *International Journal of Robotics Research (IJRR)*, 40(6-7):866–894, 2021.

Ziyu Wang, Frank Hutter, Masrour Zoghi, David Matheson, and Nando de Feitas. Bayesian optimization in a billion dimensions via random embeddings. *Journal or Artificial Intelligence Research (JAIR)*, 55:361–387, 2016b.

Eric W. Weisstein. Square root inequality. MathWorld–A Wolfram Web Resource. `http://mathworld.wolfram.com/SquareRootInequality.html`, 1999-2018.

Andrew G Wilson, Christoph Dann, Chris Lucas, and Eric P Xing. The human kernel. *Advances in Neural Information Processing Systems (NeurIPS)*, 2015.

Martin Wistuba and Josif Grabocka. Few-shot Bayesian optimization with deep kernel surrogates. In *International Conference on Learning Representations (ICLR)*, 2021.

Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. Scalable Gaussian process-based transfer surrogates for hyperparameter optimization. *Machine Learning*, 107(1):43–78, 2018.

David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

Dani Yogatama and Gideon Mann. Efficient transfer learning method for automatic hyperparameter tuning. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2014.

Guodong Zhang, Lala Li, Zachary Nado, James Martens, Sushant Sachdeva, George Dahl, Chris Shallue, and Roger B Grosse. Which algorithmic choices matter at which batch sizes? insights from a noisy quadratic model. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.