# Prior-Data Fitted Networks Can Do Mixed-Variable Bayesian Optimization

A Master's Thesis

Presented to

Eberhard-Karls-Universität Tübingen

Mathematisch-Naturwissenschaftliche Fakultät

Wilhelm-Schickard-Institut für Informatik

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Machine Learning

Timothy Shinners

March 2024

First Examiner: Dr. Katharina Eggensperger
Second Examiner: Prof. Dr. Jakob Macke

# Abstract

Bayesian optimization is an algorithm used for many black box optimization problems, such as hyperparameter optimization for machine learning models. In some settings, the black box function has a mixed variable input space that involves both numerical and categorical variables. Prior-data fitted networks (PFNs) are transformers that are trained to behave similar to Gaussian processes (GPs). They have been shown to perform well as a surrogate function in Bayesian optimization methods, offering similar performance capabilities to GPs, with reduced computational expense. However, they have not yet been applied to mixed variable settings. In this work, we train three PFNs using existing mixed variable surrogate models as priors, as well as one PFN trained on a mixture of priors. We integrate them into full mixed variable Bayesian optimization (MVBO) methods and conduct experiments on six different black box functions to assess their behavior. Findings suggest that, in MVBO settings, using trained PFNs as a surrogate function yields similar performance to that of their GP-based counterparts, while operating at a fraction of the computational expense for long optimization runs with over 500 iterations.

# Kurzfassung

Die Bayes'sche Optimierung ist ein Algorithmus, der zur Optimierung vieler Black-Box-Probleme verwendet wird, beispielsweise zur Hyperparameteroptimierung für Modelle des maschinellen Lernens. Oft verfügt die Blackbox-Funktion über einen gemischten Hyperparameterraum, der sowohl numerische als auch kategoriale Variablen umfasst. Prior-Data-Fitted-Networks (PFNs) sind Transformer, die darauf trainiert sind, sich ähnlich wie Gaußsche Prozesse (GPs) zu verhalten. Es hat sich gezeigt, dass sie als Modelle in Bayes'schen Optimierungsmethoden ähnlich gut wie GPs functioneren und Suchräume geringeren Rechenaufwand erfordern. Sie wurden jedoch noch nicht auf gemischte angewendet. In dieser Arbeit trainieren wir drei PFNs unter Verwendung vorhandener GP variationen mit gemischten Suchräume als Priors sowie einen PFN, der auf einer Mischung von Priors trainiert wird. Wir integrieren sie in MVBO-Methoden (Mixed Variable Bayesian Optimization) und führen Experimente mit sechs verschiedenen Black-Box-Funktionen durch, um ihr Verhalten zu bewerten. Die Ergebnisse deuten darauf hin, dass die Verwendung trainierter PFNs als Modelle in MVBO-Einstellungen eine ähnliche Leistung wie ihre auf GP basierenden Gegenstücke liefert und dabei nur einen Bruchteil des Rechenaufwands für lange Optimierungsläufe mit über 500 Iterationen erfordert.

# Table of Contents

# Chapter 1

# Introduction

## 1.1 Motivation

A significant amount of real-world problems can be posed as the optimization of a black box function. These functions must be optimized iteratively, due to the lack of differentiability and any prior knowledge about the behavior of the function itself. From hyperparameter optimization for machine learning models (Snoek, Larochelle, & Adams, 2012) to the optimization of a cookie recipe (Solnik et al., 2017), a huge class of problems need to be optimized via sequential experimentation to achieve desired results. Large search spaces and costly functions prevent evaluation at all possible points, so there is a need for sample efficient black box optimization methods.

Bayesian optimization (Garnett, 2023) is an iterative model-based method of black box optimization that is well suited for applications where the black box function is costly to compute. First, a probabilistic machine learning model, a so-called surrogate model, is fit to previous observations to model the black box objective. Second, an acquisition function based on the surrogate model is optimized, effectively balancing exploration and exploitation. In the case of cookie optimization (Solnik et al., 2017), the surrogate model predicts the quality of the cookies, given a specific recipe, and the acquisition function estimates the utility of attempting a proposed recipe, given the results of past attempts.

The choice of surrogate function is important to the performance of the specific BO method. Gaussian Processes have been seen as the ideal choice of surrogate model due to their flexibility, predictive performance, and reliable uncertainty estimates (Garnett, 2023). However, they do not scale well with higher numbers of observations or features. In addition, they lack flexibility, as their posterior distributions are restricted to normal distributions.

Prior-data fitted networks (Müller, Feurer, Hollmann, & Hutter, 2023), or PFNs, have been proposed as a new surrogate function. These are pre-trained networks that use in-context learning (Müller, Hollmann, Arango, Grabocka, & Hutter, 2022). In PFNs4BO (Müller et al., 2023), PFNs were trained using synthetic data sampled from a Gaussian process prior. In a single forward pass, the trained PFNs were able to observe a test and training data set, then produce outputs that accurately approximate that of its respective prior, at a fraction of the computational expense.

An important class of black box optimization problems are those involving one or more categorical variables. For example, consider the case of optimizing a cookie recipe (Solnik et al., 2017). Some variables, like "type of chocolate chip," are categorical, while others, such as amount of sugar or salt, are fully continuous across a defined range. These mixed variable input spaces present difficulties in terms of how to handle categorical variables numerically.

A variety of methods have been proposed to tackle this problem. (Dreczkowski, Grosnit, & Ammar, 2023) created a framework to implement and benchmark a number of these methods, including CoCaBO (Ru, Alvi, Nguyen, Osborne, & Roberts, 2019), Casmopolitan (Wan et al., 2021), and BODi (Deshwal et al., 2023). These methods use a Gaussian process for the surrogate function, with a mixture of numerical and categorical kernels that incorporate the mixed types of variables. However, due to the surrogate function being a Gaussian process, they suffer from the same issues as non-categorical Gaussian processes.

It would be desirable if we could use PFNs in this mixed-variable setting in order to leverage the advantages that were demonstrated in the continuous case with PFNs4BO (Müller et al. (2023)). To this end, we outline our research questions below.

## 1.2 Research Questions

The overarching research question of this work is as follows:

**Do PFNs offer a good alternative to GPs as surrogate functions in mixed-variable Bayesian optimization?**

To answer this question, we formulate the following three sub-questions:

1. Do PFNs yield similar performance to GPs as a surrogate function in mixed-variable Bayesian optimization loops?

2. Can PFN performance improve by using a mixture of priors during the PFN's

training?

3. Are PFNs more computationally efficient than their GP counterparts?

## 1.3 Contributions

In the pursuit of answering these research questions, the following contributions are laid out in this work:

1. A general method to sample the prior's hyperparameters during the PFN training procedure. This allows for easier adaptation to a wider range of priors, as it does not require manually defined prior distributions over the hyperparameters.
2. The application of Weitzman's overlap (Weitzman, 1970) to score the similarity in behavior between trained PFNs and their respective priors.
3. Three fully trained PFNs, each trained on a different prior. In addition, we also study a PFN trained on a mixture of all three priors.
4. Implementation of code that integrates PFNs into the general MCBO (Dreczkowski et al., 2023) framework, making them ready to use within a Bayesian optimization method[1].

## 1.4 Structure of This Work

In the next chapter, we cover relevant background information and related works, discussing Bayesian optimization, mixed variable Bayesian optimization, and PFNs. In section 3, we discuss the procedure used to train new PFNs, and in section 4, we explain our procedure for evaluating and comparing trained PFNs, to figure out which training parameters lead to optimal performance. In section 5, we detail our experimental procedure and present the results. We conclude with a short discussion about the results and potential future works in section 6.

[1]Implementation is planned to be released at `https://github.com/TimShinners/PFNs4MVBO`

# Chapter 2

# Background and Related Work

In this section we cover related works and background information that pertains to this work. We begin by defining black box optimization, then discuss Bayesian optimization. After that, we introduce mixed variable Bayesian optimization, and some associated methods designed for that problem. We conclude the chapter with an introduction and description of prior-data fitted networks, an alternative surrogate model to Gaussian processes.

## 2.1 Black Box Optimization

Let $X \subseteq \mathbb{R}^d$ for some $d \in \mathbb{N}$, and $f : X \to \mathbb{R}$. Black box optimization problems are a class of problems for which we want to find $x^*$ such that

$$x^* \in \mathrm{argmin}_{x \in X} f(x)$$

(Garnett, 2023). We assume the objective $f$ has no functional form, no access to the derivatives of $f$, the values $f(x)$ are potentially subject to random noise, and there is a cost associated with the computation of $f(x)$. The goal is to minimize the objective function $f$ while also minimizing the number of queries that need to be made during the optimization procedure.

## 2.2 Bayesian Optimization

Bayesian optimization is an iterative method used for black box optimization problems (Garnett, 2023) (Mockus, Tiesis, & Zilinskas, 1978) (Kushner, 1964). Given a set of recorded data from previous iterations, it yields a suggestion for the next attempt.

For each iteration, there are three components: the fitting of a surrogate function, the optimization of an acquisition function, and the update of the recorded data set, shown in Algorithm 1 (Shahriari, Swersky, Wang, Adams, & De Freitas, 2016).

**Algorithm 1** Bayesian Optimization

```
input: objective function $f$, surrogate function $\hat{m}$, acquisition function $\alpha$, initial
observations $D_1 = \{(x_i, y_i) | i \in 1, ..., k\}$, number of iterations $N$
for $n = 1, 2, 3, ..., N$ do
    fit surrogate function $\hat{m}$ to prior observations $D_n$
    select new $x_{n+1}$ by optimizing acquisition function $\alpha$

        $x_{n+1} = \text{argmax}_{x \in X} \alpha(x;\ \hat{m}, D_n)$

    query objective function $y_{n+1} = f(x_{n+1})$
    augment data $D_{n+1} \leftarrow D_n \cup (x_{n+1}, y_{n+1})$
end for
$(x^*, y^*) = (x_j, y_j)$, given $j = \text{argmax}_{y_i \in D_N}\ y_i$
return best configuration $x^*$, best observed value $y^*$
```

The surrogate function, usually some form of a Gaussian process, is fit to the recorded data. Gaussian processes can be seen as a probability distribution over functions, defined by a mean function $\mu : X \to \mathbb{R}$ and a covariance function $K : X \times X \to \mathbb{R}$ (Garnett, 2023). Let $D = \{(x_i, y_i) | i \in 1, ..., n, x \in X, y \in \mathbb{R}\}$ be a set of observations, $\mathbf{x} = \{x_i | x_i \in D\}$, $\mathbf{y} = \{y_i | y_i \in D\}$, and $\mu_{\mathbf{y}} = \frac{1}{n} \sum_{i=1}^{n} y_i$. The Gaussian process can be conditioned on the set of observations $D$. When fitting to $D$, the fitted mean and covariance functions can be calculated as:

$$\mu_D(x) = \mu(x) + K(x, \mathbf{x})\Sigma^{-1}(\mathbf{y} - \mu_{\mathbf{y}})$$

$$K_D(x, x') = K(x, x') - K(x, \mathbf{x})\Sigma^{-1}K(\mathbf{x}, x')$$

with $\Sigma = K(\mathbf{x}, \mathbf{x})$ such that $\Sigma_{ij} = K(x_i, x_j)$ (Garnett, 2023).

Given a test point $x'$ after fitting, the Gaussian process outputs a posterior distribution over the possible $y'$ values, with $y' \sim N(\mu = \mu_D(x'), \sigma^2 = K_D(x', x'))$. It should be noted that these results assume no observational noise, although it is not difficult to incorporate more general assumptions about noise into the model. For more information about Gaussian processes, we refer the reader to Gaussian processes for machine learning (Rasmussen & Williams, 2006) and

Bayesian Optimization (Garnett, 2023).

The surrogate function need not be a Gaussian process (see section 2.4), but after fitting to the recorded data, for a given input $x$, the output must be a posterior probability distribution over possible values for the target $y$. Ideally, the surrogate model will approximate the black box objective function, allowing access to its functional form, derivatives, and uncertainty estimates that we can't access through the unknown black box objective.

Next, the acquisition function estimates the utility of evaluating a given point $x$ in the next iteration. The goal is to produce high values in unexplored regions, as well as in previously explored regions that are known to yield high values from the objective $f$. By combining the surrogate function's outputs and uncertainty, the acquisition function is able to provide an efficient balance between exploration and exploitation. Prior work has introduced several acquisition functions, such as expected improvement (Mockus et al., 1978) or probability of improvement (Kushner, 1964). An example can be seen in Figure 2.1, where a Gaussian process has been fit to 5 observations, and the expected improvement has been plotted below.

The acquisition optimizer attempts to maximize the acquisition function, finding the point $x$ that yields the greatest utility at that stage in the optimization run. Unlike the objective $f$, the acquisition function is differentiable, which aids in the optimization process. A wide variety of acquisition optimizers have been proposed as well. For a more detailed introduction to Bayesian optimization, we refer the reader to Bayesian Optimization (Garnett, 2023).

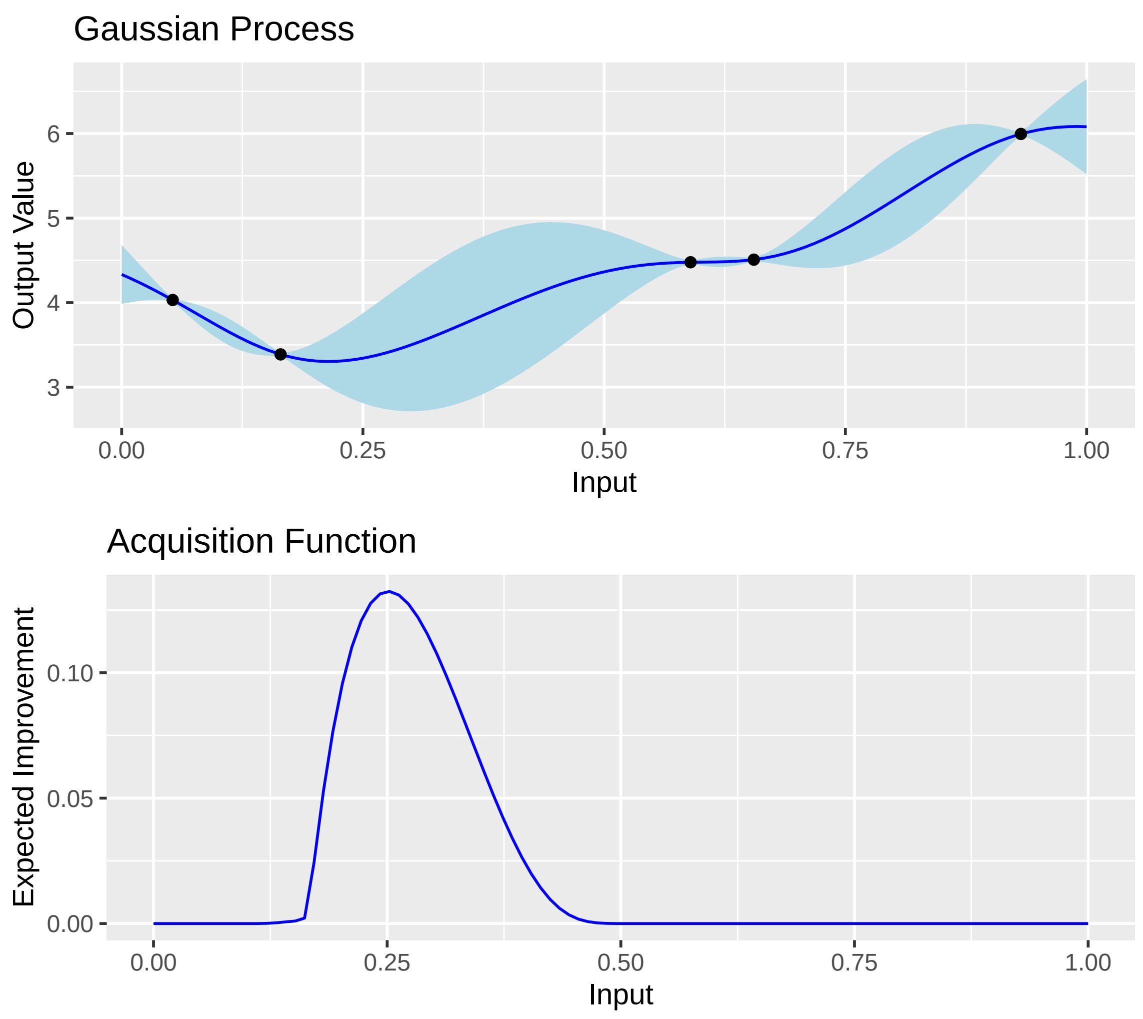

Figure 2.1: A Gaussian process, fit to 5 training points, and the corresponding values for the expected improvement

## 2.3 Mixed Variable Bayesian Optimization

Often, real-world black box optimization problems involve categorical variables as inputs. These problems have categorical or discrete variables that need to be treated differently than continuous variables. Let $X \subseteq \mathbb{R}^{d_1} \times \mathbb{N}^{d_2}$ with $d_1, d_2 \in \mathbb{N}$, and $f : x \to \mathbb{R}$. As before, we want to find $x^*$ such that

$$x^* \in \text{argmin}_{x \in X} f(x)$$

These problems have resulted in a class of mixed variable Bayesian optimization methods that can handle categorical and continuous inputs. These methods utilize surrogate functions that have been specifically designed for mixed variable tasks.

When using Gaussian processes as the surrogate function, this is usually accomplished by combining a numerical and categorical kernel. Categorical kernels are used to estimate a covariance between different categories across different categorical variables. The numerical and categorical kernels are then combined using kernel addition and multiplication to yield an output.

The optimization of the acquisition function also presents its own set of challenges, since traditional numerical optimizers cannot optimize over categorical variables. A brute force approach might be to, for each combination of categorical variables, optimize with respect to the numerical variables. However, the number of categorical combinations grows exponentially with the number of categorical dimensions, rendering this approach unfeasible for tasks with high categorical dimensionality.

In pursuit of improved mixed variable black box optimization, a variety of Bayesian optimization algorithms have been proposed. Recently, (Dreczkowski et al., 2023) implemented and conducted large scale comparisons between a number of these methods. We base our work on this study, and have chosen to focus on three methods included in this benchmark.

**CoCaBO**

CoCaBO (COntinuous and CAtegorical Bayesian Optimisation) (Ru et al., 2019) proposed a mixture of a kernel for continuous inputs $k_x(x, x')$ and a kernel for categorical inputs, $k_h(h, h')$. Faced with pros and cons of using the sum or product of the two kernels, they opted to use both, mixing the sum and product of the continuous and categorical kernels, weighted by a parameter $\lambda \in [0, 1]$:

$$k_z(z, z') = (1 - \lambda)(k_x(x, x') + k_h(h, h')) + \lambda k_x(x, x')k_h(h, h').$$

For the categorical inputs, they proposed the use of an overlap kernel. Let $c$ be the number of categorical variables, and $\sigma$ be a tunable lengthscale parameter. Then, $k_h(h, h') = \frac{\sigma}{c} \sum_{i=1}^{c} \delta(h_i, h'_i)$, where $\delta(h_i, h'_i) = 1$ if $h_i = h'_i$ and zero otherwise. For the acquisition function, they chose expected improvement, and for the acquisition optimizer they proposed to use a Multi-Armed Bandit (Ru et al., 2019) system to optimize over the categorical inputs, and a more traditional numerical optimizer for the continuous inputs.

### Casmopolitan

Casmopolitan (CAtegorical Spaces, or Mixed, OPtimisatiOn with Local-trust-regIons & TAilored Non-parametric) (Wan et al., 2021) also uses a mixture between the sum and product of a numerical and categorical kernel. For categorical inputs, however, they employ a transformed overlap kernel:

$$k_h(h, h') = \exp\left(\frac{1}{d_h}\sum_{i=1}^{d_h} l_i \delta(h_i, h'_i)\right)$$

with lengthscales $l_i$, and $d_h$ being the number of categorical dimensions in the given task. This kernel has increased flexibility over CoCaBO due to the different lengthscales that can be fitted during training. Expected improvement is used for the acquisition function, and for the acquisition optimizer, they use an interleaved search, optimizing the categorical inputs, conditioning on them, and then optimizing the continuous inputs. They also employ a trust region to prevent over-exploration during the procedure (Wan et al., 2021).

### BODi

BODi (Bayesian Optimization over High-Dimensional Combinatorial Spaces via Dictionary-based Embeddings) (Deshwal et al., 2023) uses a Hamming embedding via dictionaries. For each BO iteration, a new dictionary $\mathbf{A}$ is randomly generated, which is used to compute an embedding $\phi_{\mathbf{A}}(z)$ with the observations $\mathbf{z}$ (Deshwal et al., 2023). A GP is then fit to this embedding, which is used as the surrogate function. Similarly to the Casmopolitan (Wan et al., 2021) approach, the acquisition function is expected improvement and the acquisition optimizer is the interleaved search. BODi does not employ a trust region.

CoCaBO (Ru et al., 2019), Casmopolitan (Wan et al., 2021), and BODi (Deshwal et al., 2023) are all proposed methods for mixed-variable Bayesian Optimization. However, all three rely on Gaussian processes for the surrogate function, which, as noted earlier, do not scale well with high number of data or dimensions. They also lack flexibility in that their posterior distributions must be strictly Gaussian.

Framework and Benchmarks for Combinatorial and Mixed-variable Bayesian Optimization (MCBO) (Dreczkowski et al., 2023) introduced a modular framework for implementing a wide variety of BO methods. It allows for the selection of a surrogate function, acquisition function, and acquisition optimizer to form a BO method. They implemented a number of existing techniques, as well as some novel combinations of different BO components. They also implemented a variety of synthetic and real-world tasks for the purposes of evaluating the different methods. Throughout the rest of this work, we use MCBO's implementations of these different BO algorithms and tasks in our training procedure, evaluations, and experiments.

## 2.4 Prior-Data Fitted Networks

Prior-Data Fitted Networks are a type of transformer that have been trained to do Bayesian inference (Müller et al., 2022). In PFNs4BO (Müller et al., 2023), prior-data fitted networks (PFNs) were proposed as a surrogate model in Bayesian optimization. As input, they take a training data set with a variable number of features and observations, and a test data set containing only inputs. For each observation in the test data set, the output is a predictive distribution over the possible $y$ values. It is trained using synthetic data generated from a Prior-Data Generating Model (PDGM), with the choice of PDGM being extremely flexible.

(Müller et al., 2023) showed that trained PFNs were able to observe a training and test data set, and then produce predictive distributions that accurately approximated the behavior of its respective PDGM, all in a single forward pass, and at a fraction of the computational expense of traditional GPs. PFNs have added flexibility in the sense that they can be trained on a wide class of PDGMs, and the outputted distributions can approximate any abstract continuous probability distribution, not just Gaussians.

The architecture of a PFN is a Transformer (Vaswani et al., 2017), which consists of an encoder and a decoder. The encoder relies on an attention mechanism to encode a variable amount of inputs into a vector representation, and the decoder is able to convert that vector representation into a desired output. This allows PFNs to take training and test data sets as input, that are variable in both number of data points and number of dimensions.

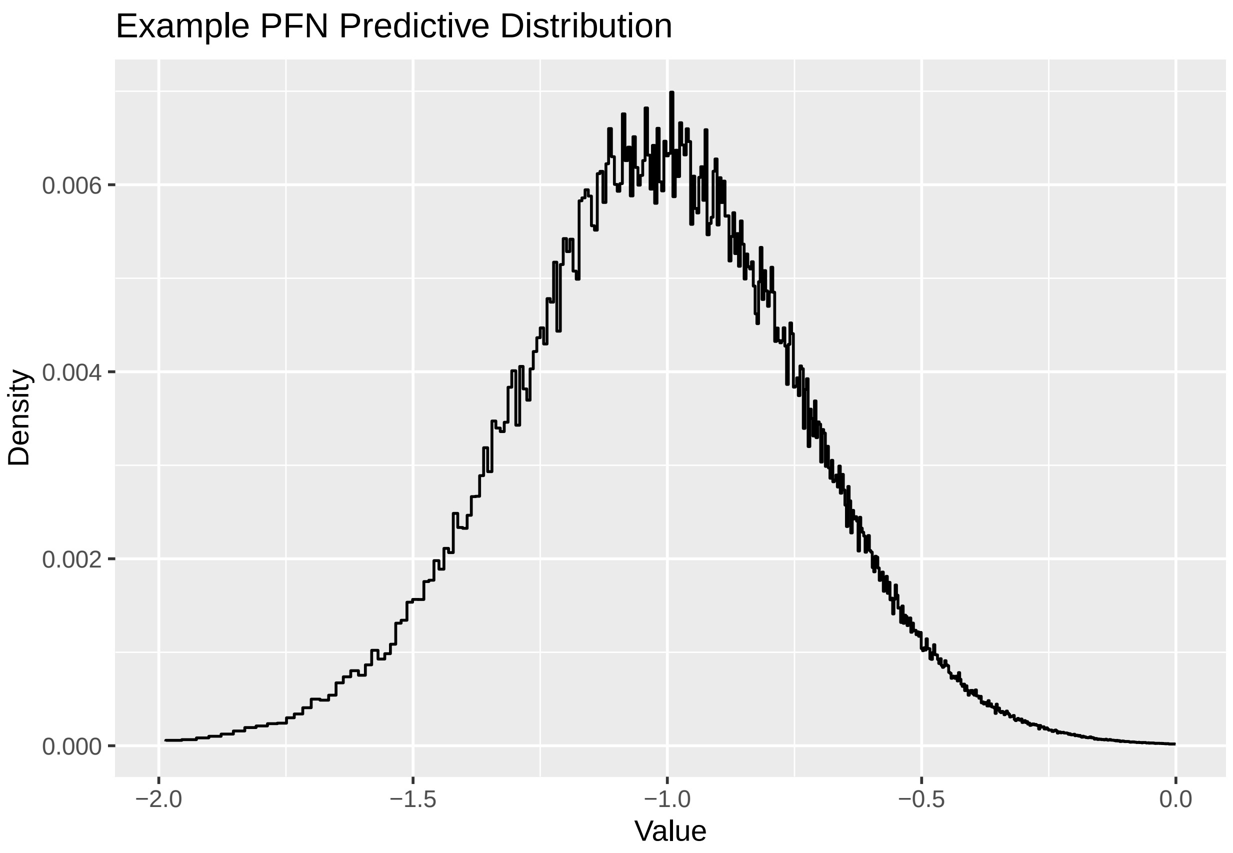

Figure 2.2: A PFN's predictive distribution for one data point

The PFN's output is a probability distribution over the possible target values. Specifically, this output is in the form of a Riemann distribution, illustrated in Figure 2.2. This is a highly flexible, discrete approximation of a continuous distribution. Riemann distributions are piecewise constant, essentially discretising the real-valued output space into a predetermined number of "buckets". The boundary of each bucket is decided upon prior to training. When making a prediction, the PFN outputs the logit value that a test $y$ point will fall into each bucket. This converts the PFN's task from regression into a classification problem, as it is made to predict which bucket a test $y$ value will fall into.

PFNs4BO (Müller et al., 2023) was able to train PFNs to successfully imitate a HEBO (Cowen-Rivers et al., 2020) PDGM. They were able to produce similar performance to HEBO in full optimization runs, at reduced computational cost. However, they focused only on continuous variables, and did not apply PFNs to mixed variable spaces with categorical variables.

# Chapter 3

# Training Methods

In this section, we discuss the methods and procedures used to train PFNs. We begin with a general overview of the training process, then discuss methods for sampling hyperparameters for the prior-data generating model.

## 3.1 Training Procedure

We closely followed the training procedure used in PFNs4BO. When training a new PFN, a Prior-Data Generating Model (PDGM) is selected, and then synthetic data sets are drawn from the PDGM, which are used as training data for the PFN. In this work, we attempted to train new PFNs using the mixed variable surrogate functions from CoCaBO (Ru et al., 2019), Casmopolitan (Wan et al., 2021), and BODi (Deshwal et al., 2023) as PDGMs.

To train a PFN, we must sample data sets from the selected PDGM. If the PDGM is a Gaussian process, then to do this, we draw $n$ values uniformly from the input space. Given these values, the covariance matrix $K$ is calculated, such that $K_{i,j} = k(x_i, x_j)$, with $k(\cdot, \cdot)$ being the GP's covariance function, and $x_i,\ x_j$ being two different points drawn from the input space. Then, $y$ values are sampled from a multivariate normal distribution $y \sim N(\mu = \mathbf{0}, \Sigma = K)$. Together, $D = \{(x_i, y_i) | i \in 1, ..., n\}$ forms one sampled data set.

When training a PFN, data sets are sampled from the PDGM and partitioned into training and test sets. The training and test sets are passed to the PFN. The cross-entropy loss between the PFN's output and the test $y$ points is calculated and used for backpropagation through the PFN.

At the start of the training procedure, the boundaries of the PFN's buckets for the Riemann distribution must be selected. First, a large number of $y$ values are drawn from the PDGM. The $y$ values are then sorted into 1000 buckets of equal population, so regions with a high density of $y$ values have smaller buckets. The boundaries of these buckets remain constant for the duration of the training procedure.

**Hyperparameters For The Prior-Data Generating Model**

All three of the PDGMs used in this work are GPs that employ a mixture of a Matern 5/2 kernel for the numerical variables, and a categorical kernel that is specific to each method. Each of these kernels have hyperparameters that influence the behavior of the synthetic data sets drawn from them, ranging from lengthscales and outputscales to noise coefficients. In order to train a PFN to behave like its PDGM, it is imperative that these hyperparameters are chosen appropriately.

In PFNs4BO (Müller et al., 2023), a PFN was trained on a HEBO (Cowen-Rivers et al., 2020) PDGM. Each hyperparameter hailed from a prior distribution, so they could be sampled efficiently and independently before generating a synthetic data set. For example, before generating each synthetic data set, the lengthscale was sampled from a $Gamma(\alpha_0, \beta_0)$ distribution and the outputscale was sampled from a $Gamma(\alpha_1, \beta_1)$ distribution. This allowed the PFN to witness a variety of different scenarios during training, which led to better performance after training.

In preliminary experiments, we found that this method did not perform well in our setup. We suspected that the prior distributions we had assigned to each hyperparameter were a poor fit. We also suspected that the hyperparameters were not independent of each other, and thus should not be sampled as such. To fix these issues, we attempted to fit the PDGM to a randomly drawn synthetic data set, then extract the PDGM's fitted hyperparameters and use them when generating synthetic data for the PFN. This guaranteed an authentic set of hyperparameters were used to generate synthetic data for the PFN, although it led to substantially increased training times.

At the start of this process, we needed to sample a random data set for the PDGM to fit to. We found that drawing a small number of $x$ values from the input space, and then drawing $y \sim N(\mu = 0, \sigma^2 = 1)$ for each point $x$ led to the best performance. The randomness of the $y$ values allowed for a wide variety of situations to occur, which in turn led to a realistic distribution of hyperparameters for the PDGM. The number of points impacted the distribution of hyperparameters. If too many points were drawn, the PDGM would act as if it was fitting to noise, becoming inflexible with high uncertainty. The hyperparameters would reflect this, and the corresponding trained PFN would become similarly inflexible. With too few points, the PDGM wouldn't have much to fit to, and so it would also become rather inflexible. As discussed in section 4.2, we found optimal performance when we increased the number of points with the number of dimensions in the input space, which led to the PFNs being flexible with calibrated and accurate predictive distributions.

## 3.2 Mixed-PDGM PFNs

After training numerous PFNs using the three PDGMs individually, we attempted to train PFNs on a mixture of those PDGMs. Each time a training data set was needed to train the PFN, one of the PDGMs was selected at random. This PDGM would then generate one entire data set on its own. The process was simple, but it ideally exposed the PFN to a wider variety of function behaviors, which would potentially improve performance in full BO runs. We trained multiple mixed-PDGM PFNs, each with different probabilities of each PDGM being selected.

# Chapter 4

# PFN Development Process

In this section, we discuss the methods used to evaluate different PFN variants. In total, we trained over 100 PFNs with a variety of different training settings. This necessitated a methodology to empirically assess the performance of PFNs and determine which variants performed better than others. We aimed for three PFNs in total, one trained using CoCaBO (Ru et al., 2019), one with Casmopolitan (Wan et al., 2021), and one with BODi (Deshwal et al., 2023). We wanted PFNs that behave similarly to their Prior-Data Generating Models (PDGMs) and perform well in regression and Bayesian optimization settings. We designed an evaluation pipeline that incorporated regression tasks and Bayesian optimization runs, as well as a direct comparison between PFNs and their respective PDGMs using an overlap score, allowing us to evaluate PFN performance across these different areas.

## 4.1 Evaluation Procedure

We assess our PFNs by measuring their performance in regression settings, and their performance as part of a BO method. In order to assess the regression capabilities of our PFNs, we composed an evaluation procedure that involved drawing training and test data sets from synthetic functions used to evaluate global optimization procedures, and calculating the test loss across a variety of different scenarios. The MCBO package (Dreczkowski et al., 2023) includes 52 synthetic functions, along with 5 real-world tasks. We opted to use 8 synthetic functions for this evaluation, chosen to include a balanced mix of smooth functions and lumpy functions with many local minima and maxima.

For these synthetic tasks, it is possible to control the number of numerical dimensions, the number of categorical dimensions, and the number of categories per categorical dimension. For evaluation, we randomly draw 100 different setups, with different numbers of dimensions and categories for each setup as follows. First, we draw the number of dimensions uniformly at random from between 2 and 18, and then the fraction of categorical dimensions. Finally, the number of categories for each categorical variable is drawn from between 2 and 10. This allowed us to observe how the PFN's performance changed with respect to the dimensionality of the task.

For each function, for each of the 100 different setups, we randomly uniformly drew a set of input points, and calculated their corresponding function values to form a training and test set. We used an increasing number of training points, $[4, 9, 16, 25, 36, 49, 64, 81, 100, 121]$, and 1000 test points to observe how the PFN's performance changed with respect to the amount of training data. For each trial, we calculated the negative log-likelihood loss and the overlap score (introduced below), averaged across the test points.

We composed another evaluation procedure to assess each PFN's performance in full BO runs. Each PFN's PDGM hailed from a different BO method. To evaluate a PFN's suitability as a BO surrogate model, we replaced the GP-based surrogate function in the original BO method with a PFN. We used the original setup as provided by the MCBO package, but altered the acquisition function settings to reduce computational costs (see Appendix B). We used the same 8 synthetic functions and procedure for drawing the number of dimensions and categories for a given setup.

For each synthetic function, for 10 different setups, and for 100 iterations, the PFN-based BO method made a suggestion and observed the result. Prior to the start of each run, the BO method was initialized with ten randomly drawn points.

### Overlap Score

To evaluate a PFN's ability to imitate the GP variant that was used to derive its PDGM. We are interested in measuring the similarity between the PFN's output probability distribution and that of its PDGM. We did not use KL-divergence because it is not symmetric and not scale-invariant. Thus, we chose to calculate Weitzman's overlap (Weitzman, 1970) between the two distributions.

Define $f_1(x)$ and $f_2(x)$ as the probability density functions for two separate distributions. Then:

$$\text{Overlap Score} = \int_{-\infty}^{\infty} \min\left(\{f_1(x), f_2(x)\}\right) dx$$

(Weitzman, 1970). This score ranges from 0 to 1, with 1 indicating that the two distributions are identical. The min() operator is symmetric, so the score itself is symmetric. It is scale invariant, and the score is easily understandable. Two examples are shown in Figure 4.1, where the area of the red shaded region corresponds to the overlap score between the two illustrated probability distributions.

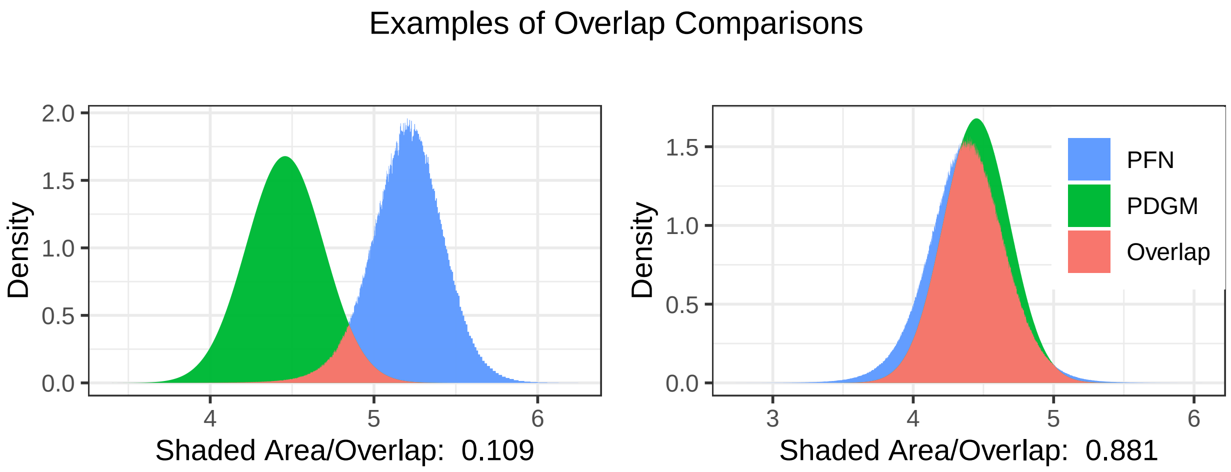

Figure 4.1: Two examples of a comparison between the output of a PFN and its PDGM. In one, the two distributions are not so similar and we see an overlap score of 0.109. In the other, the two distributions are quite similar, so we see an overlap score of 0.881.

To solve an integration problem with a min() operator, one would usually find the set of points where $f_1(x)$ and $f_2(x)$ intersect, and split the problem into piecewise integrals. However, the output of the PFN takes the form of a Riemann distribution, which made an exact solution difficult to calculate because its density function is piecewise constant. In practice, we chose to approximate this integral by calculating $\min(\{f_1(x), f_2(x)\})$ at 10,000 equidistant points, multiplying by the distance between points, and then summing across the values.

This allowed for a direct comparison between PFNs to see which behaved more like its PDGM. The GP-based PDGMs had high predictive accuracy and performed reliably in regression settings, so it followed that high overlap scores corresponded with desirable PFN behavior. The overlap score was a direct comparison between surrogate functions, but its range of $[0, 1]$ meant that it was not particularly sensitive to outliers, unlike the negative log likelihood loss. The overlap score was subject to less noise across different trials, unlike BO runs which were dependent on the initial conditions of each trial. We looked at each PFN's regression and BO performance, to investigate changes in behavior across different different training settings. However, due to the aforementioned reasons, we used each PFN's average overlap score as the main criterion by which to measure its quality.

## 4.2 Evaluation Results

In this section we discuss the specific settings that were used for each of the PFNs trained on the three different PDGMs. We begin with a short discussion of the parameters that impacted performance, then move on to a description of the specific training settings for each of the three PFNs, as well as the mixed PFN. For a discussion of training techniques that were attempted, but yielded poor PFN performance, we refer the reader to Appendix A.

Firstly, tuning the PFN's learning rate impacted performance. In PFNs4BO (Müller et al., 2023), they set the default learning rate to 0.0001. We found that doubling or tripling this value improved PFN overlap scores, but setting it any higher led to much worse performance.

We found the number of training points used to fit the PDGM's hyperparameters, discussed in Section 3.1, impacts performance of the resulting PFNs. We plot the number of training points against the average overlap score in Figure 4.2 for PFNs trained on CoCaBO's surrogate model. We found that increasing the number of points with the number of dimensions yields best results, and we observed similar results for PFNs trained on the other PDGMs.

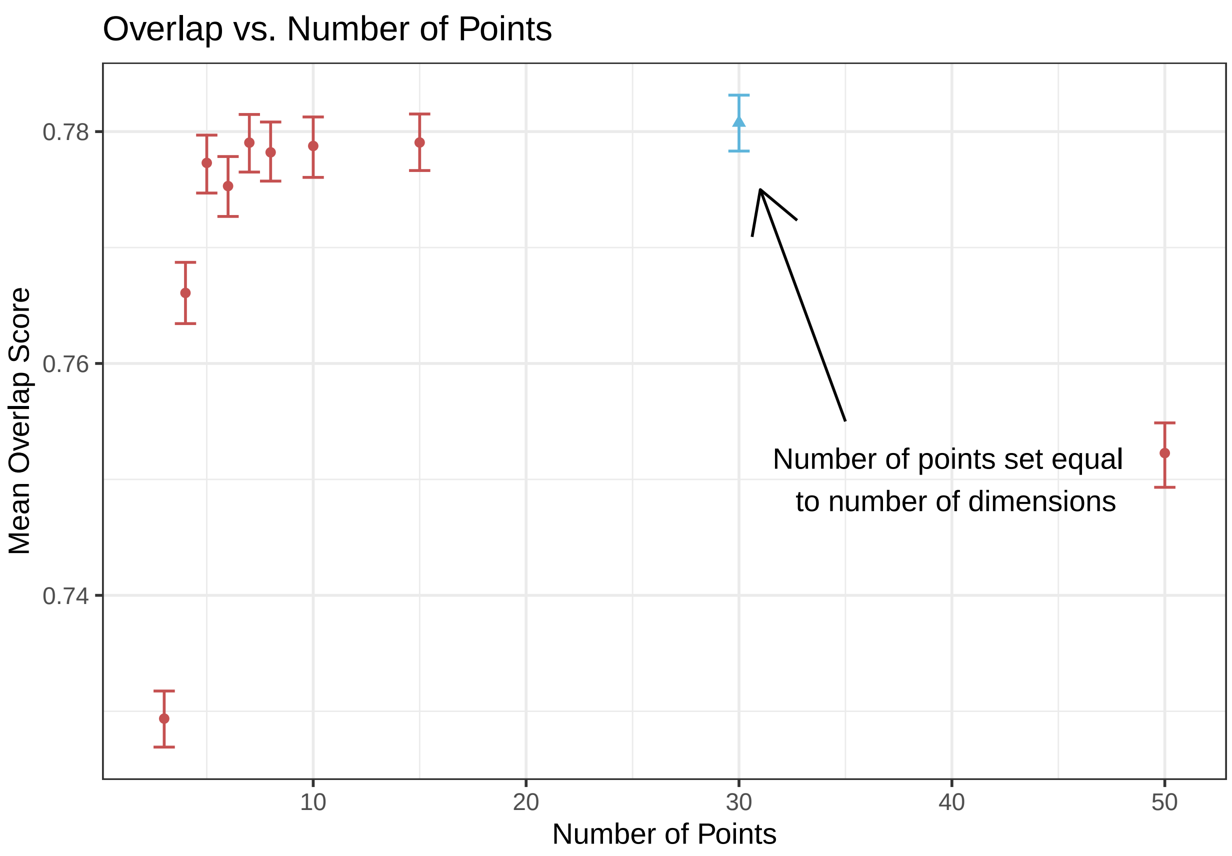

Figure 4.2: PFN overlap score plotted against the number of points drawn during training. Each point represents one PFN, and the error bars reresent 95% bootstrapped confidence intervals. We include an additional marker (blue) showing results when the value was set to increase with the number of dimensions of the input space.

We also found that the PFNs tended to yield higher uncertainties than their PDGMs, meaning that a PFN's posterior distribution tended to have a greater variance than that of its PDGM. Observational noise was one of the PDGM's tunable hyperparameters, so during training, we attempted to alter that noise with a multiplier $m \in [0, 1]$. This made the data sets sampled from the PDGM less noisy, which would in turn prevent the PFN form over-estimating its uncertainty. We found that adjusting the multiplier $m$ improved PFN performance. In Figure 4.3, we plotted the noise multiplier against the resulting PFN's average overlap score, for a set of PFNs trained on the Casmopolitan (Wan et al., 2021) PDGM. In this case, the best results were found with $m = 0.7$.

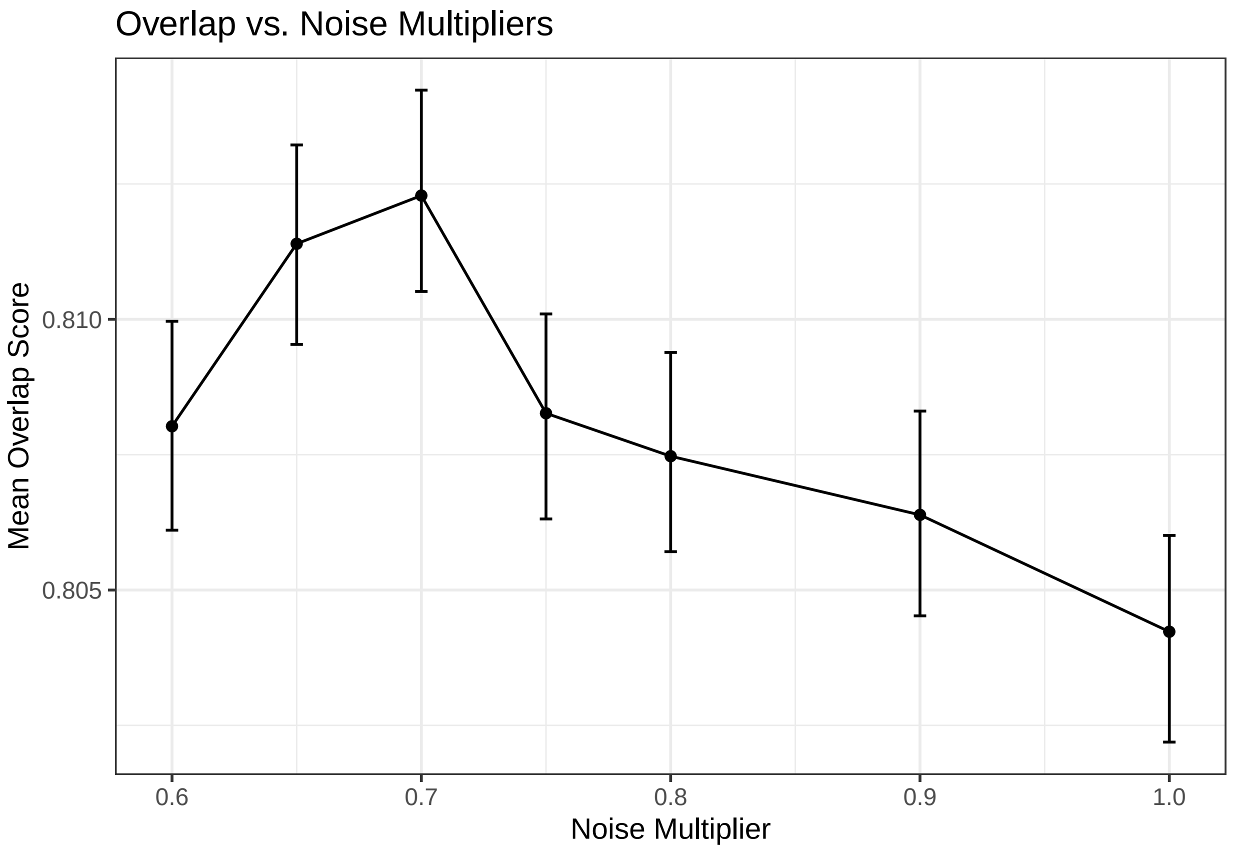

Figure 4.3: Casmopolitan-trained PFN overlap scores against the multiplier for the observational noise coefficient, with 95% confidence intervals. We did not conduct such a wide search for CoCaBO- and BODi-trained PFNs, so they are omitted from this plot.

We now discuss the training parameters that produced the best PFNs for each of the three PDGMs. The average overlap scores, as well as the specific training parameters that produced each of these PFNs are shown in Table 4.1. Points were drawn from a normal distribution for the PDGM fitting step for the CoCaBO- and Casmopolitan-trained PFNs. For the CoCaBO- and BODi-trained PFNs, the number of points to be drawn was set to the number of dimensions of the input space. For the Casmopolitan-trained PFN, this number was set to 20. Interestingly, for the BODi-trained PFNs, we found best results when using the cosine function to generate data for the PDGM fitting step (for a full explanation on this, see Appendix A). It was notably more difficult to train the BODi-trained PFNs to an acceptable level of performance. Most of the BODi-trained PFNs obtained a mean overlap score of around 0.6, while the best (and only to surpass 0.7) scored 0.771. When we attempted to train another PFN under the same settings, the average overlap score went back into the 0.6's, indicating that the best BODi-trained PFN may have been an anomaly.

Table 4.1: The training settings that produced the best PFNs for each PDGM, as well as their overlap scores.

| PFN | Overlap Score | Learning Rate | PDGM Noise Multiplier |
|---|---:|---:|---:|
| PFN-CoCaBO | 0.787 | 3e-04 | 0.5 |
| PFN-Casmopolitan | 0.812 | 2e-04 | 0.7 |
| PFN-BODi | 0.771 | 3e-04 | 1.0 |

**Mixed PFNs**

Next, we discuss the results of the mixed PFN evaluation procedure. Given that the mixed PFNs were not trained on one specific PDGM, there was no reference with which to calculate the overlap score. Thus, we relied on regression performance to compare the different mixed-PDGM PFNs. In total, 7 were trained, each with different probabilities applied to the three different PDGMs. We discuss the decision process of selecting one for further experimentation below.

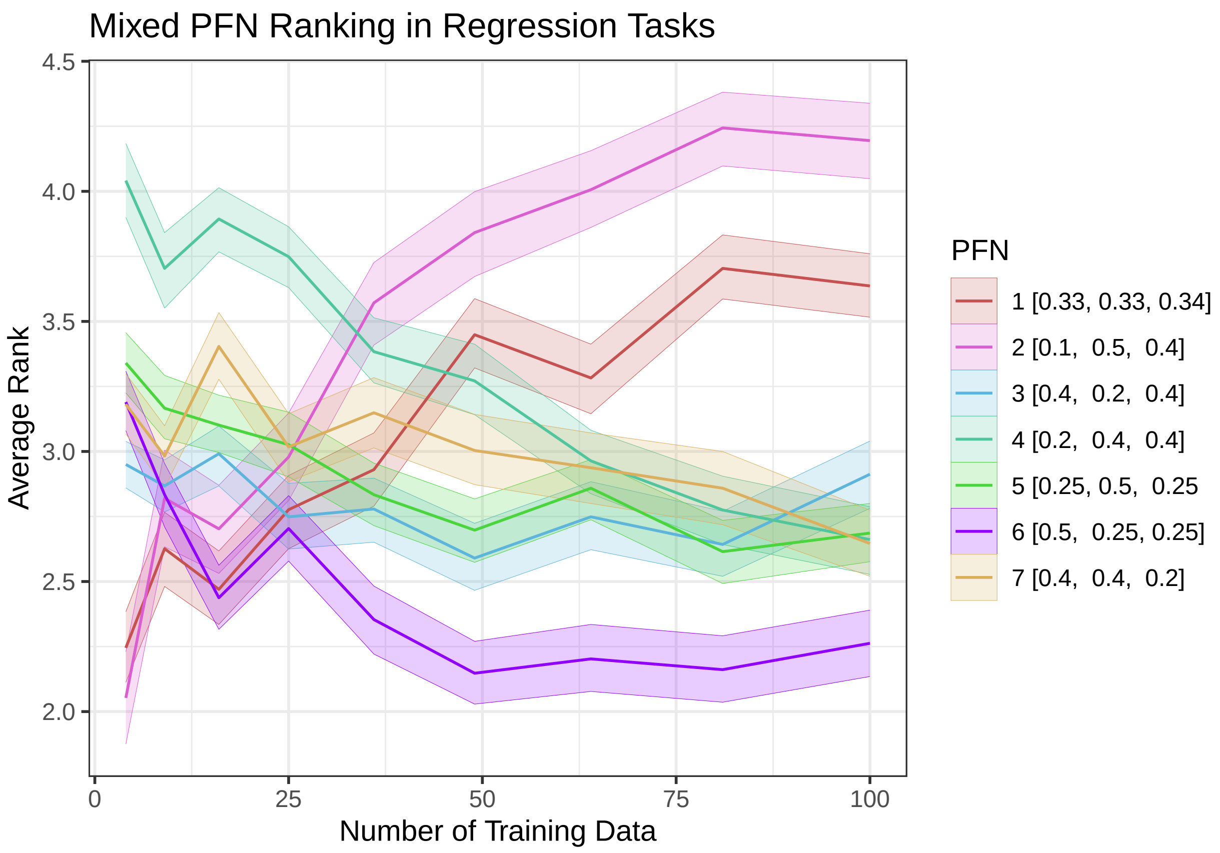

Figure 4.4: Mixed PFNs ranked by negative log likelihood loss in regression tasks, with 95% confidence intervals. For each of the seven mixed-prior PFNs, we report the probabilities used to select different PDGMs in the form [CoCaBO, Casmopolitan, BODi]

Figure 4.4 shows ranking plots for the differed mixed PFNs in regression tasks. The rankings are with respect to the negative log-likelihood loss. For each setup where a PFN was given a training and test set, and made to make predictions, the negative log-likelihood was calculated and averaged across all test points. Then, this loss was used to calculate rankings across each of the PFNs. These ranks were then averaged across all scenarios with a given amount of training data, which is shown in the plot. The 6th PFN that we trained is clearly superior to the others in regression tasks. This PFN was trained on a mixture where CoCaBO had a probability of 0.5, Casmopolitan had a probability of 0.25, and BODi had a probability of 0.25.

# Chapter 5

# Experiments

In this section, we begin with an outline of the experimental procedure used to test the different trained PFNs. Then, we present the results of the experiments, with a discussion of how these results answer the different research questions in Section 1.2.

## 5.1 Procedure

In this section we describe the experimental procedure used to evaluate the performance of the trained PFNs, compared to that of their respective Prior-Data Generating Models (PDGMs).

For experiments, one real world task and five synthetic functions were chosen to evaluate the performance of different surrogate functions in regression and BO settings. Like the evaluation procedure in section 4, the five synthetic functions were chosen to include a mixture of behaviors. We illustrate these functions in Figure 5.1, for the setting with one continuous variable and one categorical variable with three categories. The synthetic functions can be defined with any number of numerical and categorical dimensions, making it possible for the task dimensionality to be randomly sampled in the experiments. The real-world task was to optimize XGBoost hyperparameters, which involved four categorical variables and four continuous variables. None of these tasks were used during development described in Section 4. During training, we limited the maximum dimensionality of the PFNs to 18 (following PFNs4BO (Müller et al., 2023)), so we could not use the rest of the real-world tasks included in the MCBO framework (Dreczkowski et al., 2023).

To evaluate the overlap and regression performances of our trained PFNs, we followed the same procedure as in section 4, except the tasks were switched out with the new ones chosen for our experiments. Due to the large number of function queries per experiment, only the synthetic functions were used in the regression experiments. The XGBoost hyperparameter optimization task took multiple seconds per function call, rendering overlap and regression experiments unfeasible with this task. Since the BO experiments were set to run for 200 iterations, we also added extra regression trials with greater amounts of training data, using all square numbers up to 200. In addition to testing each PFN and its PDGM, we also tested a “dummy” model, which, for all inputs, simply predicted the mean and variance of the training data. This acted as a baseline for comparisons among the other methods.

For experiments involving full Bayesian optimization runs, for each task, 30 optimization runs were completed, each with a different set of initial conditions. Like 4, for the synthetic tasks, the total number of dimensions was drawn uniformly between 2 and 18. That number was randomly split to define the number of numerical and categorical dimensions. The number of categories in each categorical dimension was uniformly drawn from between 2 and 10. Before the start of each optimization run, ten initial observations were drawn randomly from the input space. After the initial observations, each optimization run lasted for 200 iterations. As a baseline for the BO experiments, we included a random search, which suggested a randomly drawn point at each iteration. We recorded the time taken for each method to record an observation and make a suggestion at each iteration.

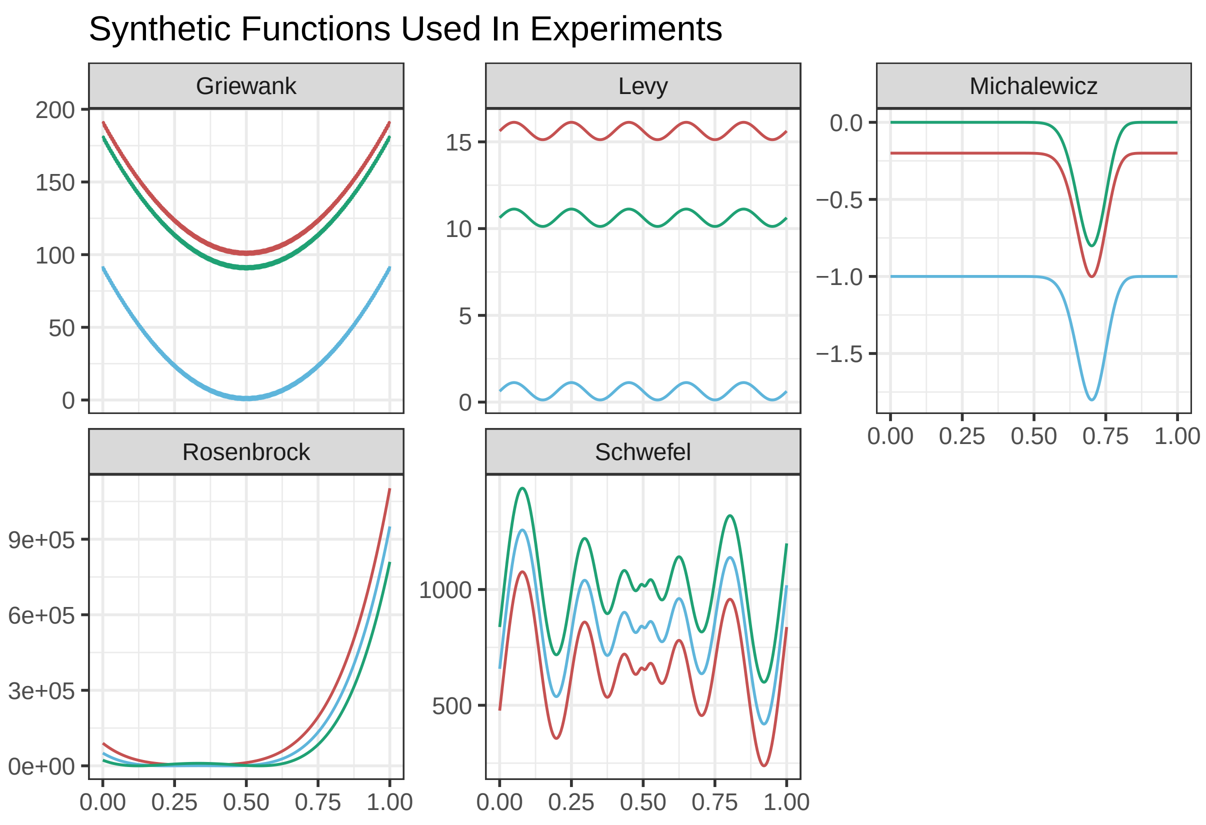

Figure 5.1: The different synthetic functions used in experiments, with one numerical variable and one categorical variable with three categories.

As was done in Section 4.1, we altered the acquisition optimizer settings for the BO experiments to reduce computation times. For specific information on these alterations, we refer the reader to Appendix B.

The experiments were conducted on a compute node consisting of 2 Intel Xeon Gold 6240 processors and 8 Nvidia RTX 2800ti GPUs.

## 5.2 Results

In this section, we use experimental results to answer the research questions proposed in Section 1.2. We begin with the first sub-question in Section 5.2.1, presenting results from the experiments outlined in Section 5.1. We answer the second sub-question in Section 5.2.2 with an analysis of the mixed-PDGM PFN's experimental performance. We present empirically observed computation costs for each optimizer in Section 5.2.3, to answer the third sub-question. We conclude the chapter with a discussion of the main research question in Section 5.2.4.

### 5.2.1 Do PFNs yield similar performance to GPs as a surrogate function in mixed-variable Bayesian optimization loops?

In this section we present experimental results, and discuss whether PFNs yield competitive performance with GPs as a surrogate function in mixed-variable Bayesian optimization methods. We present results from the regression experiments, covering the regression performance and overlap scores of each PFN. After that, the results for the BO experiments are presented.

#### Regression

Figure 5.2 shows the average rank of each model, averaged across all tasks and setups. The rankings are calculated with respect to the negative log likelihood loss, but we found the results to be similar when using the mean squared error. In these regression experiments, we found that the PFNs outperform their respective PDGMs for small amounts of training data, roughly less than 50. As the amount of training data increases, the GPs begin to outperform the PFNs. For the Schwefel and Michalewicz functions, the CoCaBO- and Casmopolitan-trained PFNs outperform all of the GP-based models. For examples of PFNs and their PDGMs fitting to individual training data sets, see Appendix C.

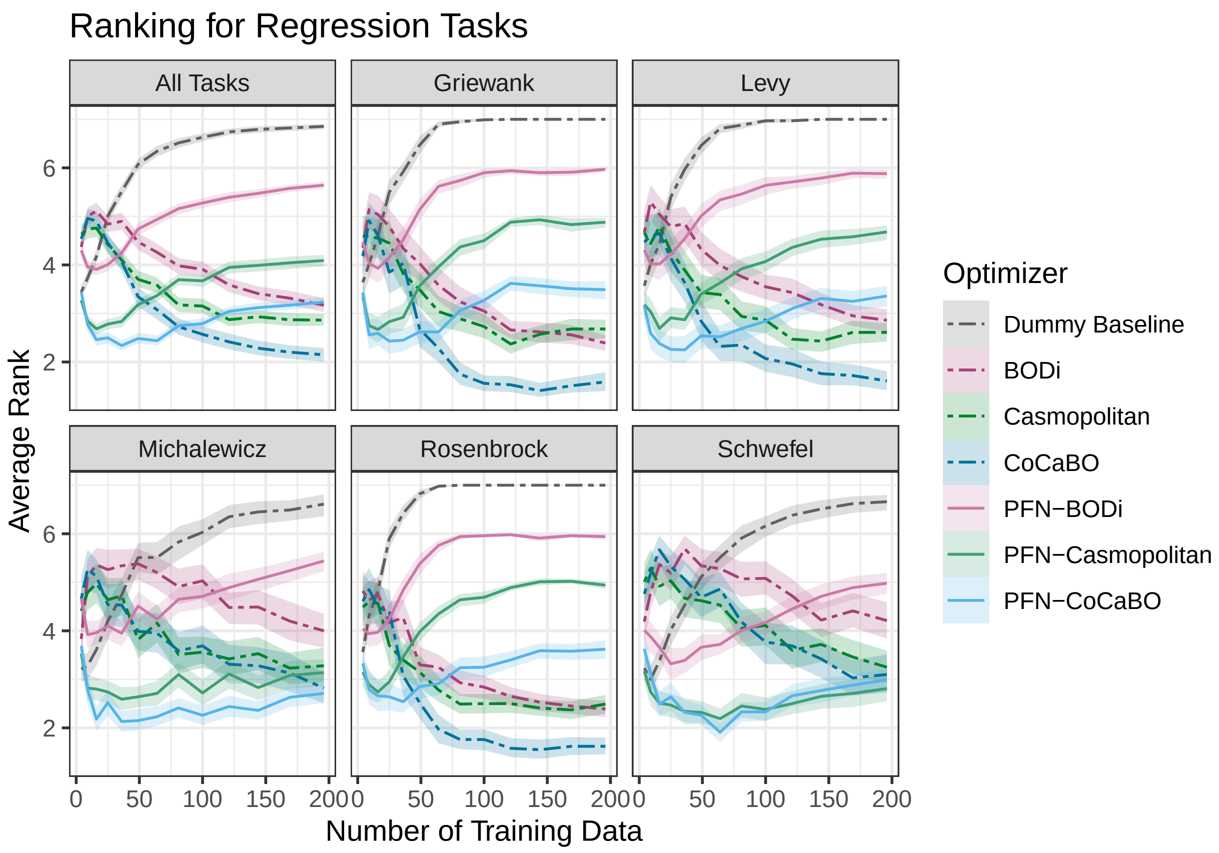

Figure 5.2: Ranks of different models, with respect to negative log likelihood across different objective functions. 95% bootstrapped confidence intervals are illustrated by the shaded regions.

### Overlap

In our experiments, averaged across all trials, we found the CoCaBO-trained PFN to have an average overlap of 0.797, with a 95% confidence interval of $[0.795, 0.799]$. For the Casmopolitan-trained PFN, the average overlap was 0.816, with a 95% confidence interval of $[0.815, 0.818]$. For the BODi-trained PFN, the average overlap was 0.752, with a 95% confidence interval of $[0.750, 0.755]$. These values are closely aligned with the average overlaps calculated in the evaluation stage, and suggest that our models are adequate at approximating the behavior of their PDGMs.

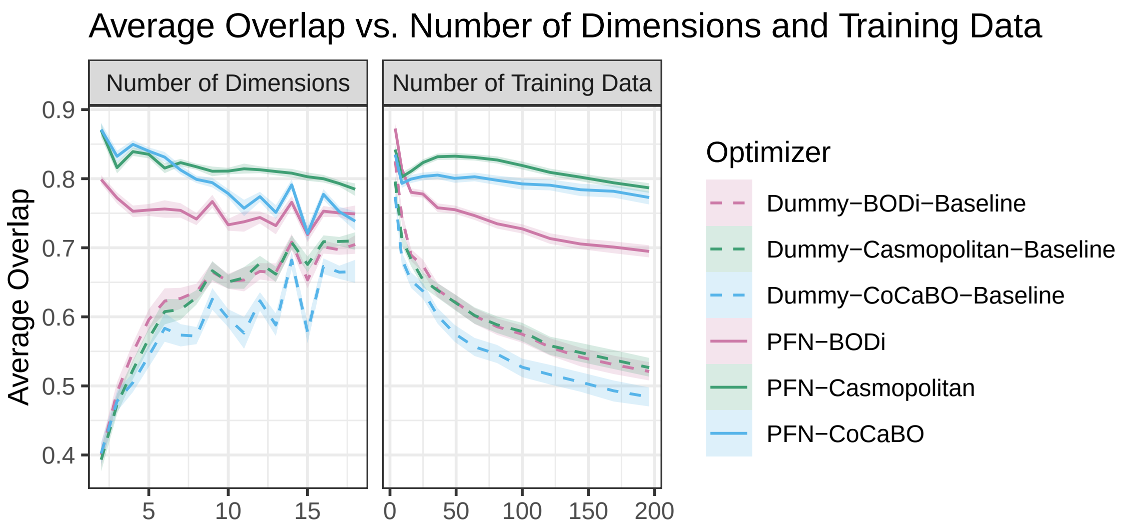

Figure 5.3: Overlap with respect to number of dimensions and training data, averaged across all setups, with 95% confidence intervals.

Figure 5.3 shows the overlap scores of each of the three PFNs, averaged across the number of training data in one plot, and number of dimensions in the other. 95% confidence intervals are shown in the shaded regions. Also included in the plot are the overlap scores achieved by the "dummy model" which predicts the training mean and variance for all test points. The overlap scores of the PFNs tend to decrease slightly with more training data and more dimensions.

It is noteworthy that the "dummy" model performs better in scenarios with low data and high dimensionality. This corresponds to scenarios with a high expected distance to the nearest training point, meaning that the GPs would suffer from higher epistemic uncertainty. This would lead them to behave similarly to the "dummy" model for large portions of the input space.

## BO

We now present and discuss the results of the BO experiments. We begin with a general comparison of all the methods included in the experiments. Then, we will compare the performance of each PFN against its respective GP-based counterpart.

The rank of each optimizer at each iteration, averaged across all tasks and all setups, is shown in Figure 5.4. As would be expected, the random baseline performs worst. Casmopolitan and the Casmopolitan-trained PFN are both strong contenders, roughly tied for best overall method.

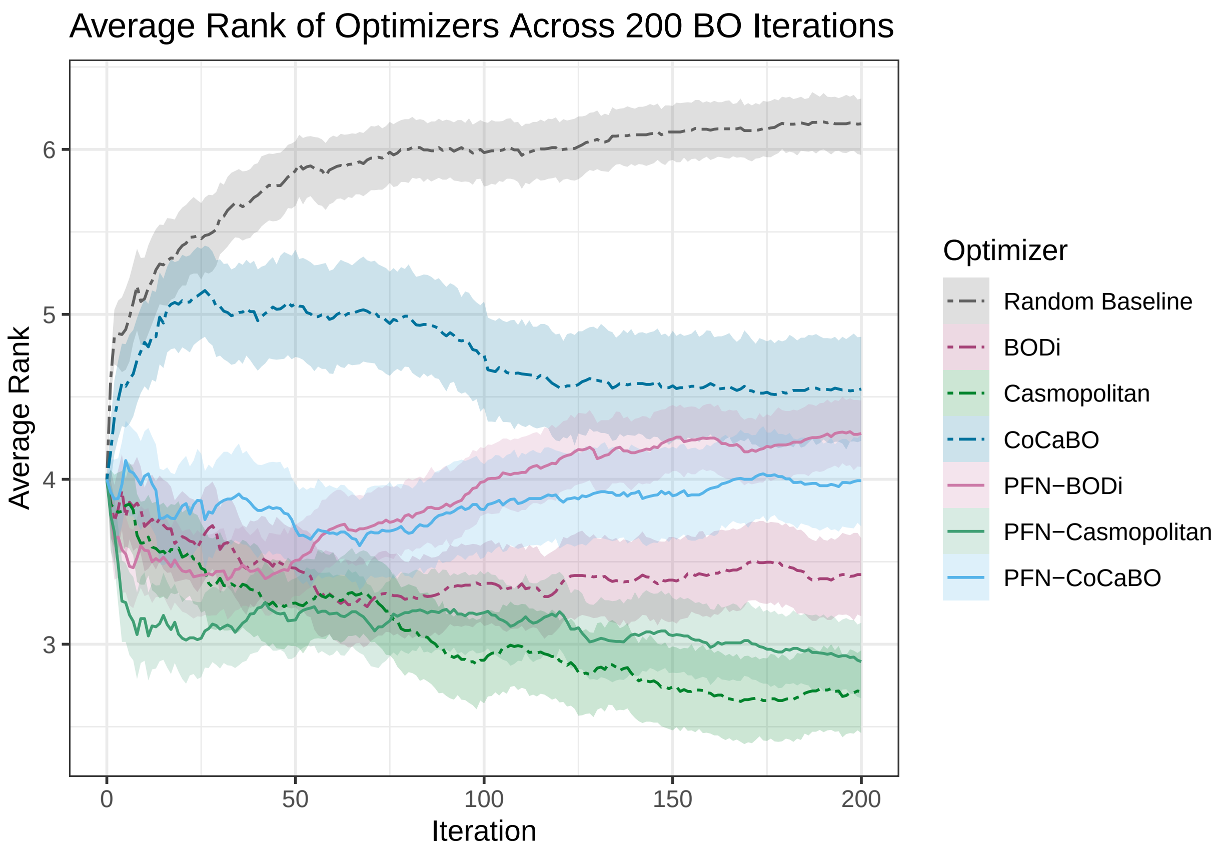

Figure 5.4: Ranks of different optimizers with respect to iteration in BO run, averaged across all tasks and all runs. The shaded regions represent the 95% bootstrapped confidence intervals around each point.

The ranks in Figure 5.4 are averaged across all of the tasks used in the experiments. The rankings varied considerably from task to task, so conclusions drawn from these experiments may be subject to the specific experimental conditions, specifically the selection of tasks. A discussion on the differences in optimizer performance across tasks, as well as task-specific ranking plots, can be found in Appendix D.

We remark that the Casmopolitan-trained PFN performs best out of all the tested methods in the opening iterations. This aligns with the PFNs' superior performance in regression settings discussed above. For the later iterations, the Casmopolitan-trained PFN shares a similar rank to the Casmopolitan method, and obtains the second-best average rank at the final iteration. Also, the CoCaBO-trained PFN clearly outperforms the original CoCaBO method, suggesting that switching a GP-based surrogate function to a PFN can yield direct improvements to the BO method. The BODI-trained PFN underperformed compared to the original BODi method, but this result is very task dependent, as shown in Figures 5.5 and 5.6.

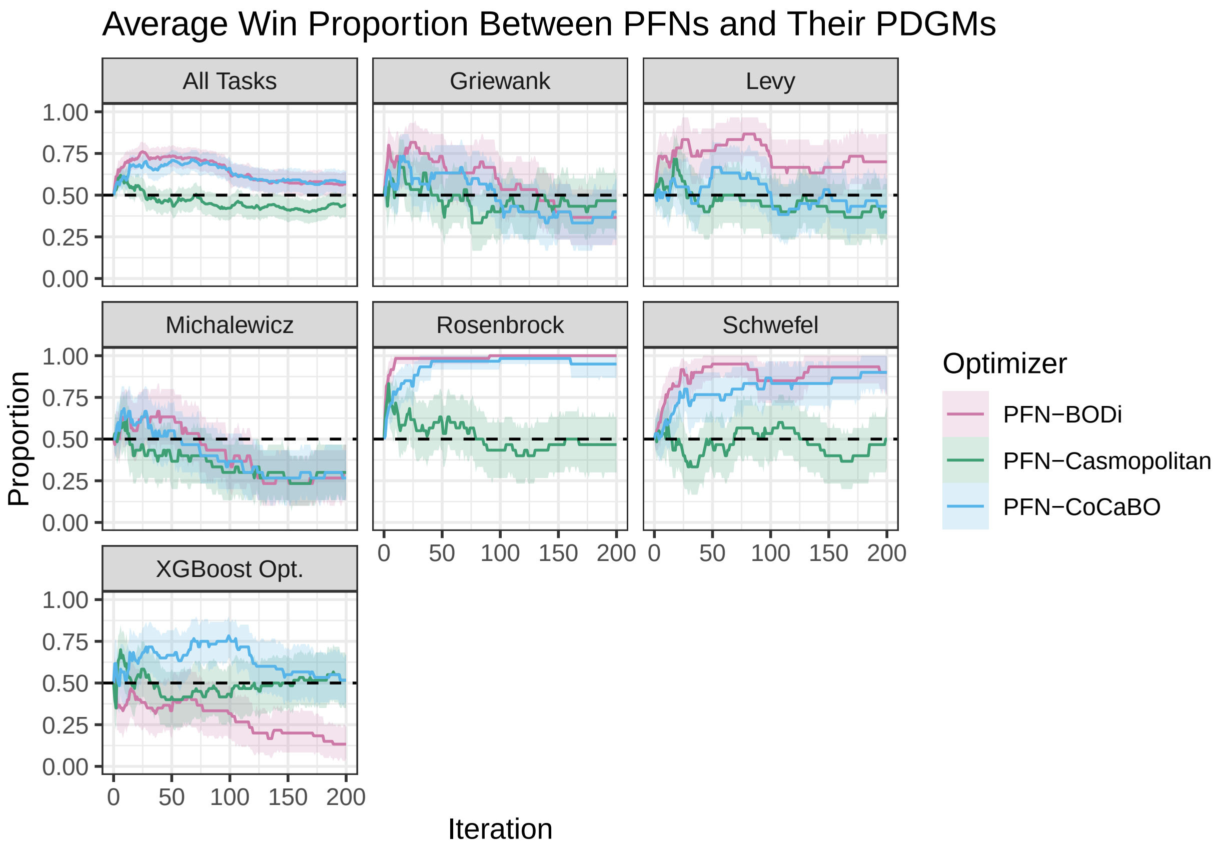

Figure 5.5: Proportion of trials where the PFN's best y value is better than that of its PDGM, for a given iteration and task, averaged across the different setups. The shaded regions represent the 95% bootstrapped confidence interval for each point.

We now look more closely at one-on-one comparisons between the PFNs and their respective GP-based methods. For each task and each iteration, we calculated the proportion of trials where the PFN's best obtained value was superior to that of its respective GP-based method. The results are shown in Figure 5.5. Points greater than 0.5 at a given iteration indicate that, in more than half of the trials, the PFN was ahead of its respective GP-based method.

At the 200-th and final iteration, the proportion of trials in which the CoCaBO-trained PFN found a better point than its GP-based counterpart was 0.58, with a 95% confidence interval of $[0.51, 0.65]$. For the Casmopolitan-trained PFN, this proportion was 0.44, with an interval of $[0.37, 0.51]$. For the BODi-trained PFN, this proportion was 0.57, with an interval of $[0.50, 0.64]$.

Generally, across all tasks, the proportions remain close to or slightly greater than 0.5, indicating an advantage when using PFNs as the surrogate model. The GP-based methods performed very poorly on the Rosenbrock function, yielding extremely favorable results for the PFN-based methods. The BODi-trained PFN outperforms BODi in most cases, with the XGBoost optimization task being the only function where the BODi-trained PFN performed substantially worse.

We note that the results in Figures 5.4 and 5.5 need not agree. For example, suppose optimizers A and B are compared across ten trials. In four trials, they have overall ranks 1 and 7, while in the six remaining trials, they have ranks 7 and 6, respectively. The overall average ranks would suggest that optimizer A is superior, while a one-on-one comparison would suggest superiority to optimizer B. To offer another angle of comparison, we investigate the average differences between the best obtained values at each iteration, shown in Figure 5.6.

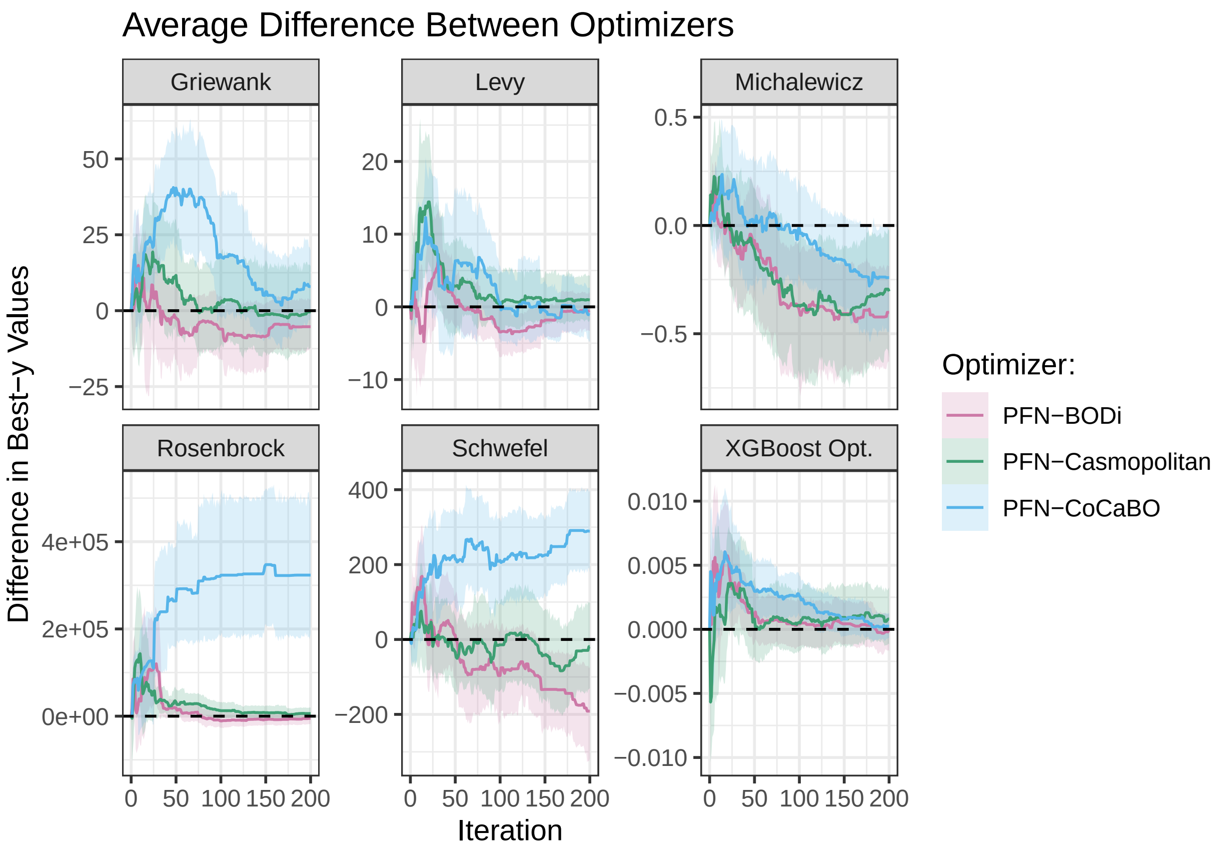

Figure 5.6: Each optimizer's best obtained value was recorded at each iteration of each trial. We calculated the difference between the best obtained values from each PFN- and respective GP-based method, for each iteration and each task, averaged across all trials. The shaded regions represent the 95% bootstrapped confidence intervals for each point. In the plots, positive values imply the PFN-based method is better than its GP-based counterpart.

For each task and each iteration, we calculated the difference in best obtained values between each PFN and its GP-based counterpart, averaged across all setups. The results are shown in Figure 5.6. Points greater than zero imply that, on average, the best value found by the PFN-based method was better than that of its PDGM. This information is different from that in Figure 5.5 because, rather than simply counting the “wins” of each method, Figure 5.6 takes the magnitude of each “win” into account. We see that, for the most part, the PFNs perform somewhat similarly to their respective GP-based methods. The CoCaBO-trained PFN outperforms CoCaBO in almost every instance, while BODi and Casmopolitan outperform their respective PFN-based methods when optimizing the Schwefel and Michalewicz functions. We note the disagreement between Figures 5.5 and 5.6 on the Rosenbrock function. Figure 5.5 suggests that the BODI- and Casmopolitan-trained PFNs were superior to their respective PDGMs, but they are shown to be nearly even in Figure 5.6.

Across Figures 5.4, 5.5, and 5.6, there is evidence to suggest that the PFNs' performance in a BO setting is similar, if not better, than that of a GP. Figure 5.4 portrayed the Casmopolitan-trained PFN as a contender for best overall method included in the experiments. In Figure 5.6, we found that the best obtained value by PFN-based methods, on average, was superior to that obtained by their respective GP-based methods. Overall, the PFNs display proficiency as a surrogate function that rivals that of their GP-based counterparts.

### 5.2.2 Can PFN performance improve by using a mixture of PDGMs during the PFN's training?

We now attempt to determine if the mixed-PDGM PFN yields better performance over the other PFNs. As discussed in Section 4.2, one mixed-PDGM PFN was included in the regression and BO experiments. In this section, we discuss the performance of the mixed-PDGM PFN, and determine whether it yields improved performance over the other methods. For the BO experiments, we needed to choose an acquisition function and acquisition optimizer for the mixed-PDGM PFN. Initial tests showed that the expected improvement and the interleaved search acquisition optimizer provided best results, so this is the configuration that we used for the BO experiments.

In Figures 5.7 and 5.8, we make one comparison between the mixed-PDGM PFN and the GP-based methods, as well as one comparison between the mixed-PDGM PFN and the other PFNs.

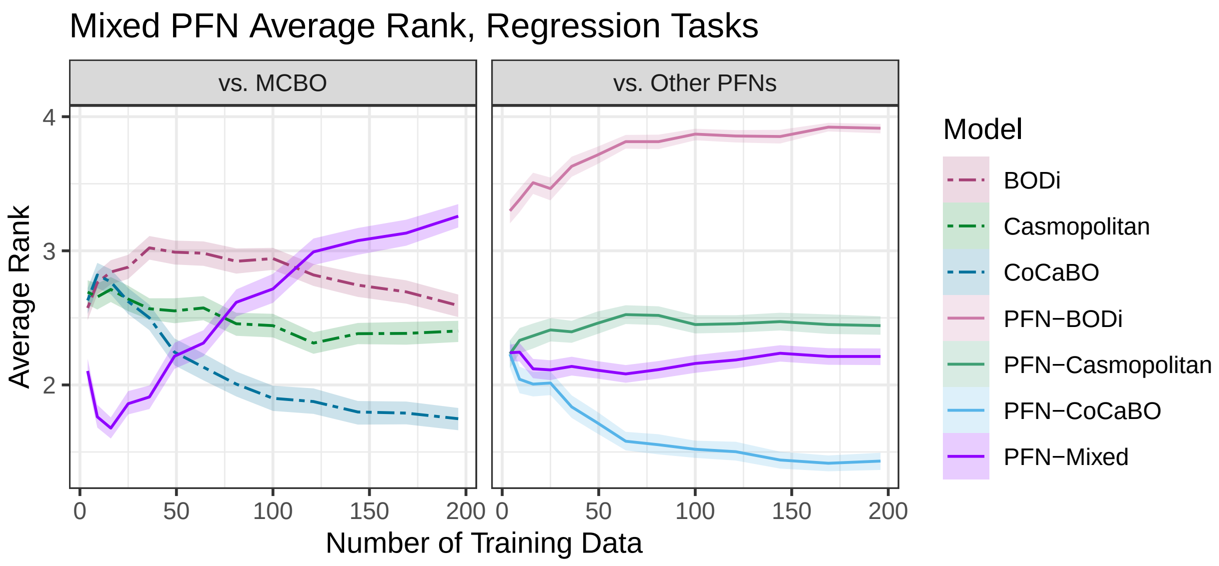

Figure 5.7: The average rank of the mixed-PDGM PFN plotted against the number of training data, with respect to negative log likelihood loss in a regression setting.

The results of the regression experiments, shown in Figure 5.7, illustrate that the performance of the mixed-PDGM PFN is similar to the other PFNs when compared to the GP-based models. The mixed-PDGM PFN appears superior for smaller amounts of training data, and inferior when the amount of training data roughly exceeds 50. When compared to the other PFNs, we find that the mixed-PDGM PFN does not exhibit improved performance over the existing PFNs. Specifically, for regression tasks it seems that the CoCaBO-trained PFN remains superior.

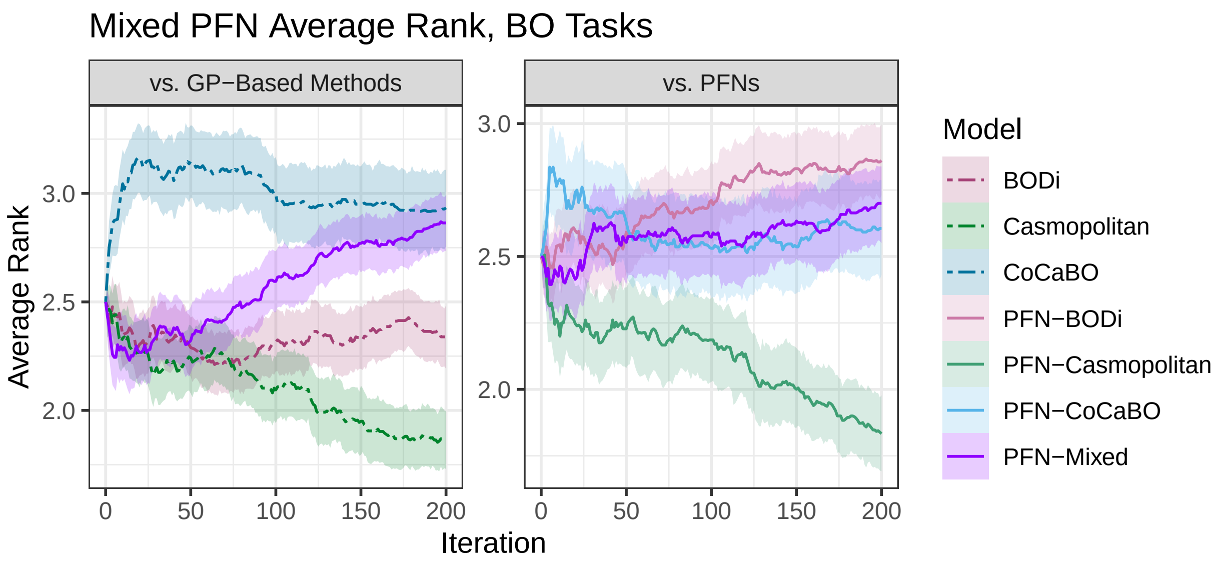

Figure 5.8: The average rank of the mixed-PDGM PFN in full BO runs. 95% bootstrapped confidence intervals are illustrated by the shaded regions.

Figure 5.8 shows the average rank of the mixed-PDGM PFN when compared to GP-based methods and the other PFN methods, in the BO experiments. Once again, the mixed-PDGM PFN does not appear superior to the other methods in a meaningful way. There was not a single individual task where the mixed-PDGM PFN exhibited superior performance to the other PFNs, so this result was not particularly dependent on the tasks included in the experiment. We conclude that we were unable to yield improvements by training a PFN on a mixture of PDGMs.

### 5.2.3 Are PFNs more computationally efficient than their GP counterparts?

We now compare computation times between GP- and PFN-based methods. For each iteration of the BO experiments, we recorded the time taken to make one observation and one suggestion. The resulting times, plotted against the number of iterations for each optimizer, is shown in the left half of Figure 5.9. The values are averaged across all tasks and all setups, and the 95% confidence intervals are illustrated by the shaded regions. The results show that the PFNs were, for the most part, slower than their GP counterparts. We note the CoCaBO-trained PFN is faster than the other two, this difference is caused by its computationally cheaper acquisition optimizer compared to that of the Casmopolitan- or BODi-trained PFNs.

The results do not fulfill the promise of drastically decreased computation times for PFNs, although it is clear that the computation time per iteration increases more rapidly with the GP-based methods. To investigate this further, we set up another test to see if the GP-based methods would become slower after more observations. For 30 repetitions, each optimizer was given 99 randomly drawn observations, before making one timed observation, and one timed suggestion. For this test, we only used one synthetic black box function, as we were not interested in the optimizers' performance, but the time taken to observe and suggest with different amounts of previous observations. The results of this test are shown in the right half of Figure 5.9.

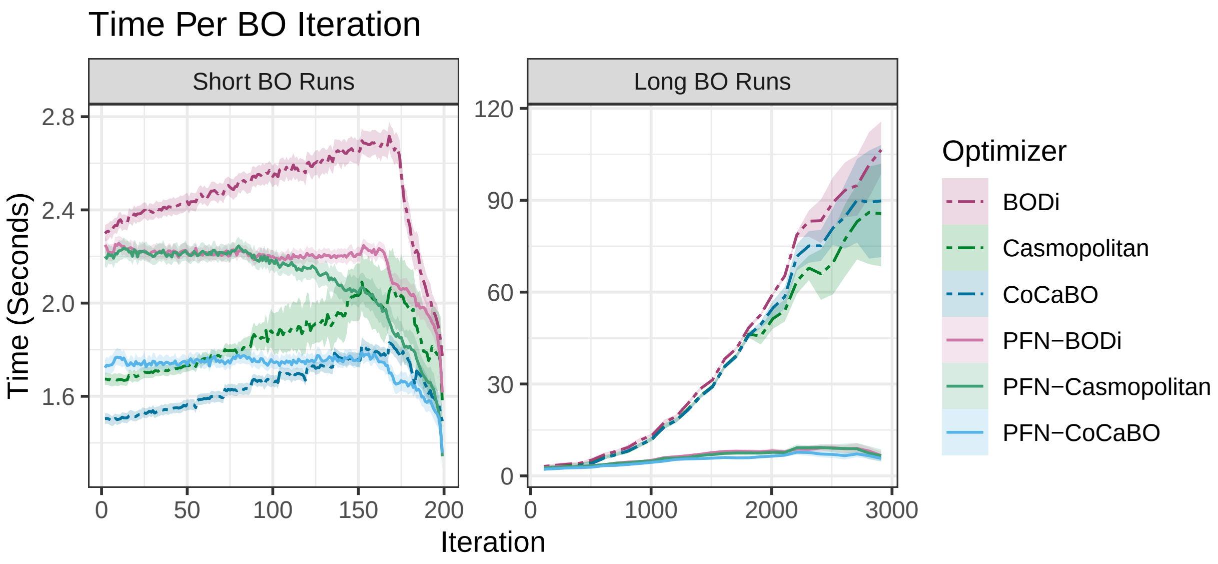

Figure 5.9: Time taken for an optimizer to make one observation and one suggestion, with 95% confidence intervals.

The right half of Figure 5.9 illustrates a stark contrast between GP-based methods and PFN-based methods, with the PFN-based methods holding a clear advantage over GP-based methods in terms of computational expense. This distinction only becomes apparent after the 500th iteration, but beyond that, there is a clear advantage to using PFNs as a surrogate function in a BO method, in that the computation time per iteration is magnitudes less than that of a GP surrogate function.

### 5.2.4 Do PFNs offer a good alternative to GPs as surrogate functions in mixed-variable Bayesian optimization?

Our experimental results provide evidence that PFNs offer a good alternative to GPs as the surrogate function in a BO method. In Section 5.2.1, we found that PFNs perform competitively with their GP-based counterparts. In regression settings, we found PFNs to be superior to GPs for small amounts of training data. We found that our PFNs scored an average overlap of 0.797, 0.816, and 0.752 for the CoCaBO-, Casmopolitan- and BODi-trained PFNs, respectively. This indicates a strong similarity in behavior for each PFN and its respective PDGM. The BO experiments also suggest competitive performance between PFN- and GP-based methods, with the Casmopolitan-trained PFN contending for the best overall method included in the experiments.

In Section 5.2.2, we did not find substantial improvements with the mixed-PDGM PFN. It performed worse than the Casmopolitan-trained PFN in almost every setup.

In Section 5.2.3, findings showed definitively that PFNs are computationally cheaper than their GP counterparts when the number of previously recorded observations is large. This illustrates a clear advantage for the use of PFNs as a surrogate model over traditional GPs.

Given the competitive performance of PFNs when compared to GPs in BO settings, and their vastly reduced computational expense, we conclude that PFNs do offer a good alternative to GPs as a surrogate model in mixed variable BO methods.

# Chapter 6

# Conclusion

PFNs have been shown to perform well as a surrogate function in fully continuous BO methods. Real-world problems, such as hyperparameter tuning for machine learning models, often involve categorical dimensions. In this work, we studied how to extend PFNs for these mixed variable problems.

We outlined a general method to sample the PDGM's hyperparameters during the PFN training procedure. We applied Weitzman's overlap (Weitzman, 1970) to score the similarity between a PFN and its PDGM during regression tasks. We trained three PFNs on three different PDGMs, as well as one PFN trained on a mixture of the three. In Section 5.2, we used our implementation of a PFN-based BO method to conduct experiments, determining the usefulness of the trained PFNs.

In our experiments, findings suggested that similar performance could be expected after switching a BO method's surrogate function from a GP to a PFN, while holding the acquisition function and acquisition optimizer constant. In regression settings, the PFNs outperformed their respective GPs for small amounts of data. In BO settings, the PFNs were competitive, with the Casmopolitan-trained PFN nearly achieving the best rank out of all the methods included in the experiments. At the termination of the BO runs, the proportion of trials in which PFN-based methods were ahead of their GP-based counterparts was 0.58, 0.44, and 0.57, for the CoCaBO-PFN, Casmopolitan-PFN, and BODi-PFN, respectively. The results were similarly close when calculating the average difference in best suggestions between the PFNs and their GP-based counterparts. The overarching theme of the results suggest that, in reference to research question 1.1, the PFNs exhibit similar quality in performance when compared to GPs as a surrogate function in a BO setting.

Training a PFN on a mixture of PDGMs did not yield improvements over the other PFNs. In a regression setting, the mixed-PDGM PFN performed similarly to the other PFNs when compared to the GP-based models, and it ranked second-best when compared to the other PFNs. In BO settings, the mixed-PDGM PFN did not exhibit substantial superiority over any of the other PFNs. With regard to research question 1.2, we conclude that we were unable to improve PFN performance when training on a mixture of PDGMs.

Lastly, in Section 5.2.3, we showed that, for BO runs lasting longer than 500 iterations, there is a distinct advantage to using PFNs in terms of computational expense. While this difference was not observable in the opening iterations, after enough observations have been recorded it becomes clear and obvious that the PFN-based methods observe and suggest much faster than GPs in a BO loop. With regard to research question 1.3, PFNs hold a substantial advantage over GPs in terms of computational expense.

Given their similar performances in BO runs, and the substantial reduction in computational expense for PFNs, we conclude that our PFNs provide a beneficial alternative to GPs as the surrogate function in a mixed variable BO setting.

## Future Works

There are a number of directions to go with future work in this area. Firstly, it should not be difficult to increase the maximum number of dimensions that the PFNs can take as input. The maximum number of dimensions is a tunable parameter that must be selected prior to the start of training. In PFNs4BO (Müller et al., 2023), it was set to 18, and in following their training procedure, we opted to leave it at 18. Increasing this parameter would lead to increased computational expense during training, but it would also generalize the PFN's applicability to a wider class of mixed variable objective functions. Some other training parameters may need to be retuned, but besides that this would likely make for a very simple extension of this work.

Also during the training procedure, to generate training data for the PFN we sampled batches of data from the PDGM, and then split these batches into a training and test set. An alternative would be to decouple the sampling process for the training and test sets. One could draw the training data from some predetermined source (like a different Gaussian process, or a class of functions like polynomials), then fit the PDGM to that training data and sample the test data from the PDGM. This would likely lead to increased training times, since one batch of data could not be split multiple times. However, it would eliminate all issues regarding the PDGM's hyperparameter distributions, since the hyperparameters would be defined during the fitting process. This would allow the PFN to learn how the PDGM behaves with respect to different sources of training data as well. An implementation of this idea would likely lead to a drastic increase in the PFN's overlap score for a given PDGM.

Lastly, in our evaluation and experiments, we did not conduct any tests with noisy observations. If random noise were to be added to the objective functions during the evaluation and experimentation of the PFNs, the results may come out differently. It may also be interesting to assess PFN performance with respect to the amount of noise added to the objective.

# Appendix A

# Failed PFN Training Techniques

In this section, we discuss PFN training techniques that were attempted, but resulted in poor PFN performance.

First, to sample the PDGM's hyperparameters described in Section 3, we needed a way to generate small data sets for the PDGM to fit to. We draw a set of input $x$ points and sample targets $y \sim N(\mu = 0, \sigma^2 = 1)$. Instead of directly sampling $y$, we used $y = \cos(a^T x)$, with the elements of $a$ being drawn uniformly from the range $[0, 7]$. However, this led to PFNs that were very inflexible and performed very poorly, with the exception of the BODi-trained PFN.

Second, we attempted different categorical encodings, which is required to pass data to the PFN. By default, the numerical variables are scaled to the range $[0, 1]$, and categorical variables are represented with integer values $\{0, 1, ...\}$. If the input space contained a binary categorical variable, the PFN would only see the values 0 and 1, making it potentially difficult to discern whether it was a categorical variable or numeric. We also found it potentially problematic that the categorical values were on such a different scale. The PFNs were not compatible with the high dimensionality that resulted from one-hot encodings. We attempted three different categorical encodings: adding 2 to each categorical variable (a binary categorical variable would now have the values $\{2, 3\}$ instead of $\{0, 1\}$), rescaling the categorical values to the range $[0, 1]$, and combining these two ideas, rescaling and then adding 2, so that all categorical values were rescaled to the range $[2, 3]$. All of the PFNs trained using any of these categorical encodings performed very poorly, so we used the default encoding.

Third, we tried to manually rescale the buckets of the PFN's Riemann distribution. The PFN's Riemann distributions tended to have very high resolution near its center, with that resolution decreasing further out. The thought was, with manually rescaled buckets, the needlessly high resolution towards the center could be traded for higher resolution further out, giving the PFN more predictive ability for points with low likelihood. We attempted to scale the PFN's buckets with a variety of different coefficients. All of these efforts led to inferior performance from the PFNs.

It is not particularly clear why all of these ideas failed to improve PFN performance. We anticipated certain problems that the PFN might encounter, and it is conceivable that the aforementioned ideas are all solutions to problems that might not exist in the first place.

# Appendix B

# Acquisition Optimizer Settings

In initial BO trials, we found that the default settings for the MCBO (Dreczkowski et al., 2023) implementations of different acquisition optimizers led to prohibitively long computation times to produce a suggestion. Specifically, in this work, we focused on the Multi-Armed Bandit (MAB) (Ru et al., 2019) and Interleaved Search (IS) (Wan et al., 2021) acquisition optimizers. For MAB, we identified one parameter, and for IS, two parameters that could be altered to decrease the optimizer's computational budget. We created a test so that we could investigate the trade-off between time and performance, allowing us to bring the computation times to a viable level while only incurring a small, acceptable drawback in optimizer performance.

For the test, we chose four synthetic tasks from the MCBO (Dreczkowski et al., 2023) package. For each task, a number of numerical and categorical dimensions was randomly drawn. Then, we initialized a BO loop. We initialized two acquisition optimizers, one with the default settings, and one with the "faster" settings. At each iteration, both acquisition optimizers produced a suggestion. The $y$ values for both suggestions were calculated and recorded. The "faster" suggestion was used for the observation in the next iteration of the loop. Each BO loop went on for 100 iterations.

We did this test for a variety of different settings and configurations. After the tests concluded, the $y$ values were rescaled for each task such that

$$\mathbf{y}_{\text{rescaled}} = \frac{\mathbf{y} - y_{\min}}{y_0 - y_{\min}}$$

where $\mathbf{y}$ is an array of all $y$ values recorded for a given task, $y_{\min}$ is the minimum value across all optimization runs for that task, and $y_0$ is the best $y$ value prior to the optimization run ($y_0$ comes from the randomly drawn initial observations). While not all of the $y$ values necessarily fell into the range $[0, 1]$, it provided a rough way to standardize the acquisition optimizers' suggestions across different tasks.

After this rescaling, we calculated the difference between the $y$ value associated with the "fast" suggestion and the "default" suggestion, and we averaged this difference across all tasks and iterations. This gave us a measure for the sacrifice in performance when switching to "faster" settings. The results of these tests are shown in Figures B.1 and B.2. It is worth noting that, when the test was run with default settings, the average difference did not perfectly come out to 0. We believe there may have been some leaked randomness somewhere in the test, and we note that while the differences were not perfectly 0, they were fairly close.

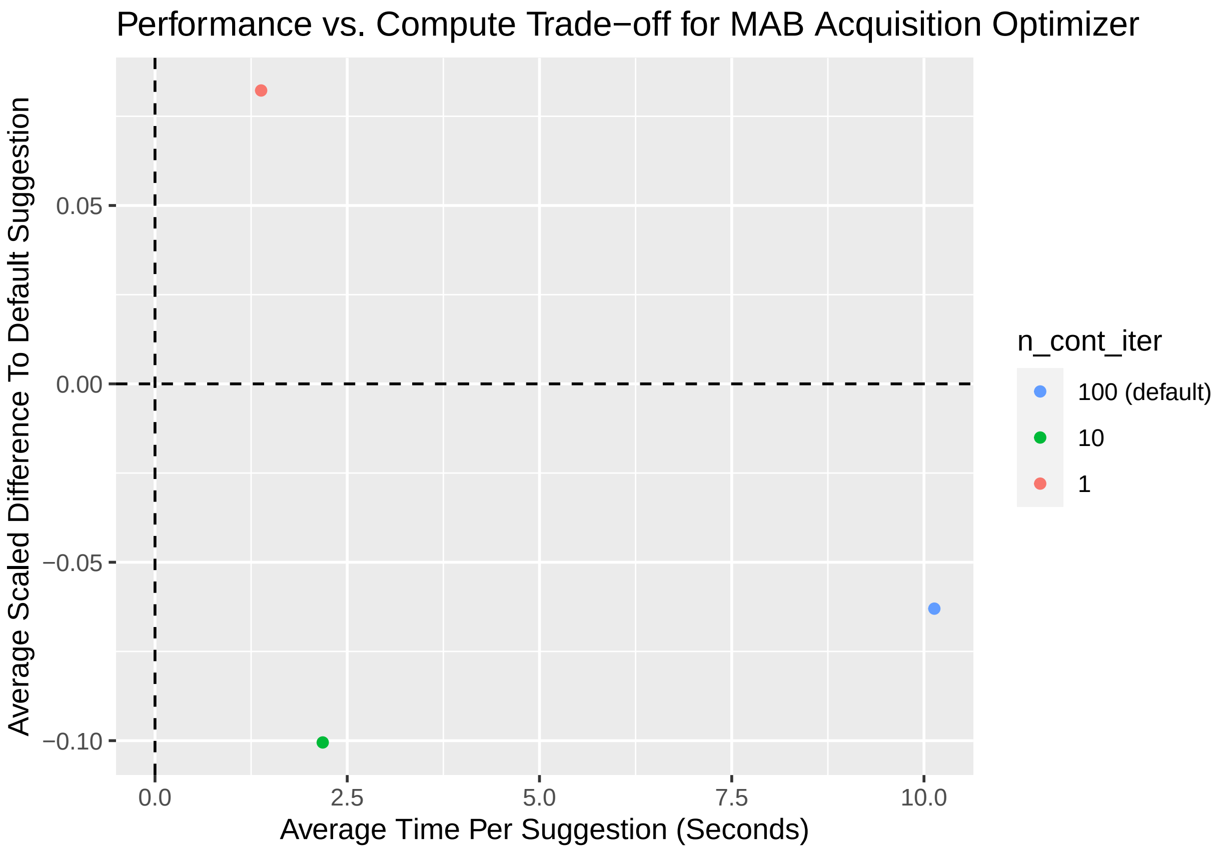

Figure B.1: The performance vs. compute trade-off for the MAB acquisition optimizer, when changing the n_cont_iter setting between values 1, 10, and 100 (default)

In Figure B.1, the average time per suggestion is plotted against the average difference in $y$ value between the faster settings and the default settings for the MAB optimizer. From this test, we concluded that setting the `n_cont_iter` parameter to 10 maintained a high quality of suggestions, while decreasing the average computation time per suggestion from over 10 seconds to less than 2.5. For this reason, we set the value to 10 throughout all of our evaluations and experiments.

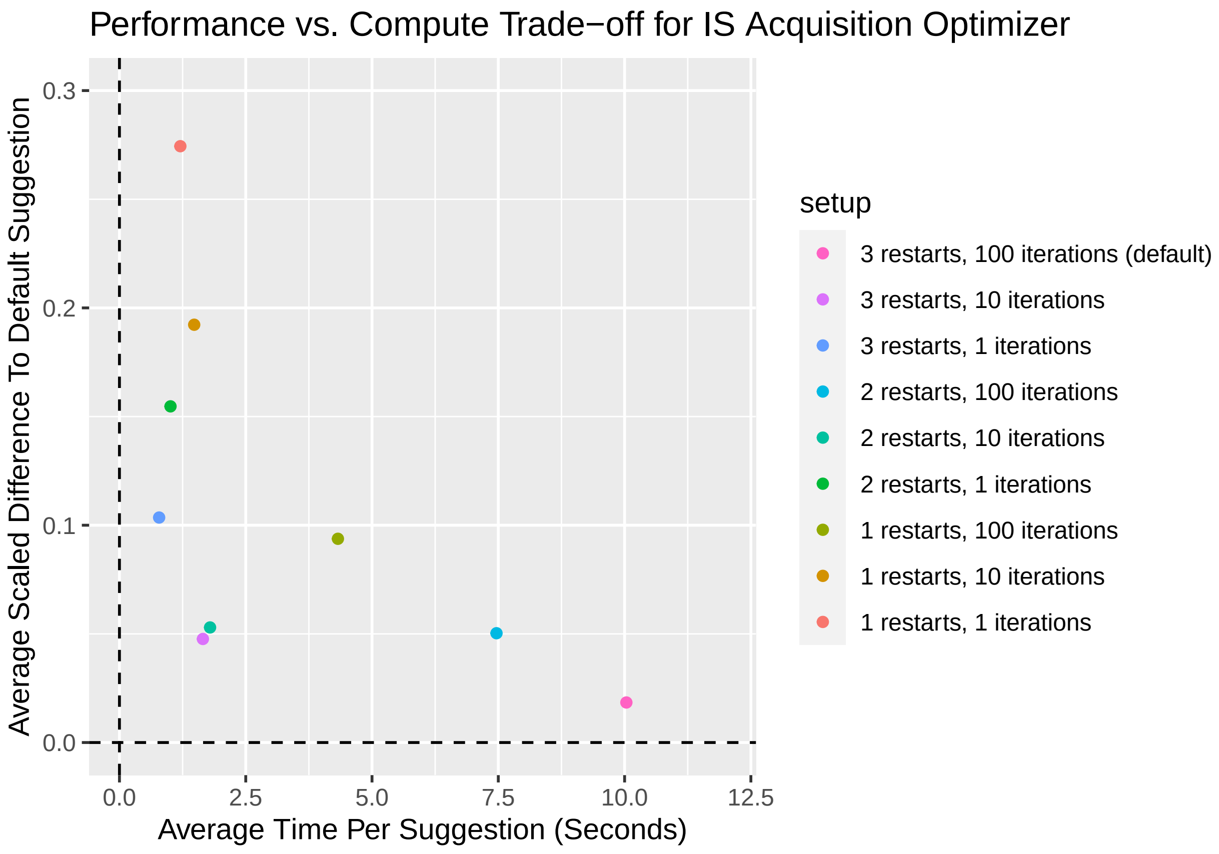

Figure B.2: The performance vs. compute trade-off for the IS acquisition optimizer, when changing the n_restarts and n_iter settings across a range of values.

As done previously, the average time per suggestion is plotted against the average difference in $y$ value in Figure B.2, this time with respect to different settings for the IS optimizer. From this test, we decided that the optimal configuration involved setting the number of restarts to 2, and the number of optimization iterations to 10. This new configuration yielded suggestions that were, on average, only 0.05 worse than suggestions from the default configuration. At the same time, the average computation time decreased to roughly $\frac{1}{5}$ that of the default configuration. We deemed this to be an acceptable trade-off between computation time and performance, so we used this configuration with the IS optimizer throughout our evaluations and experiments.

# Appendix C

# PFN Behavior in a Regression Setting

This section is a small investigation into the behavior of different models when fit to training data. We fit each model to training data sets of different sizes, drawn from the Ackley function with one continuous variable and one binary categorical variable. Figures C.1, C.2, and C.3 show how the PFNs, as well as their respective PDGMs, fit to different amounts of training data. The surrogate functions from BODi, Casmopolitan, and CoCaBO differ only in their categorical kernels, and there is only one binary categorical variable in this setting, so their behavior appears extremely similar across Figures C.1, C.2, and C.3.

We note that the GP-based models have an occasional tendency to fit very inflexibly to training data. This behavior can be seen in the setting with 5 training points in the plots below. It usually only occurs in settings with less training data, and it may be a contributing factor as to why the PFNs performed better in this setting in the regression experiments in Section 5.2, since the PFNs do not exhibit this behavior. Aside from this, we note the general similarity between each PFN and its PDGM. For the most part, there are no major deviations between each PFN and its PDGM.

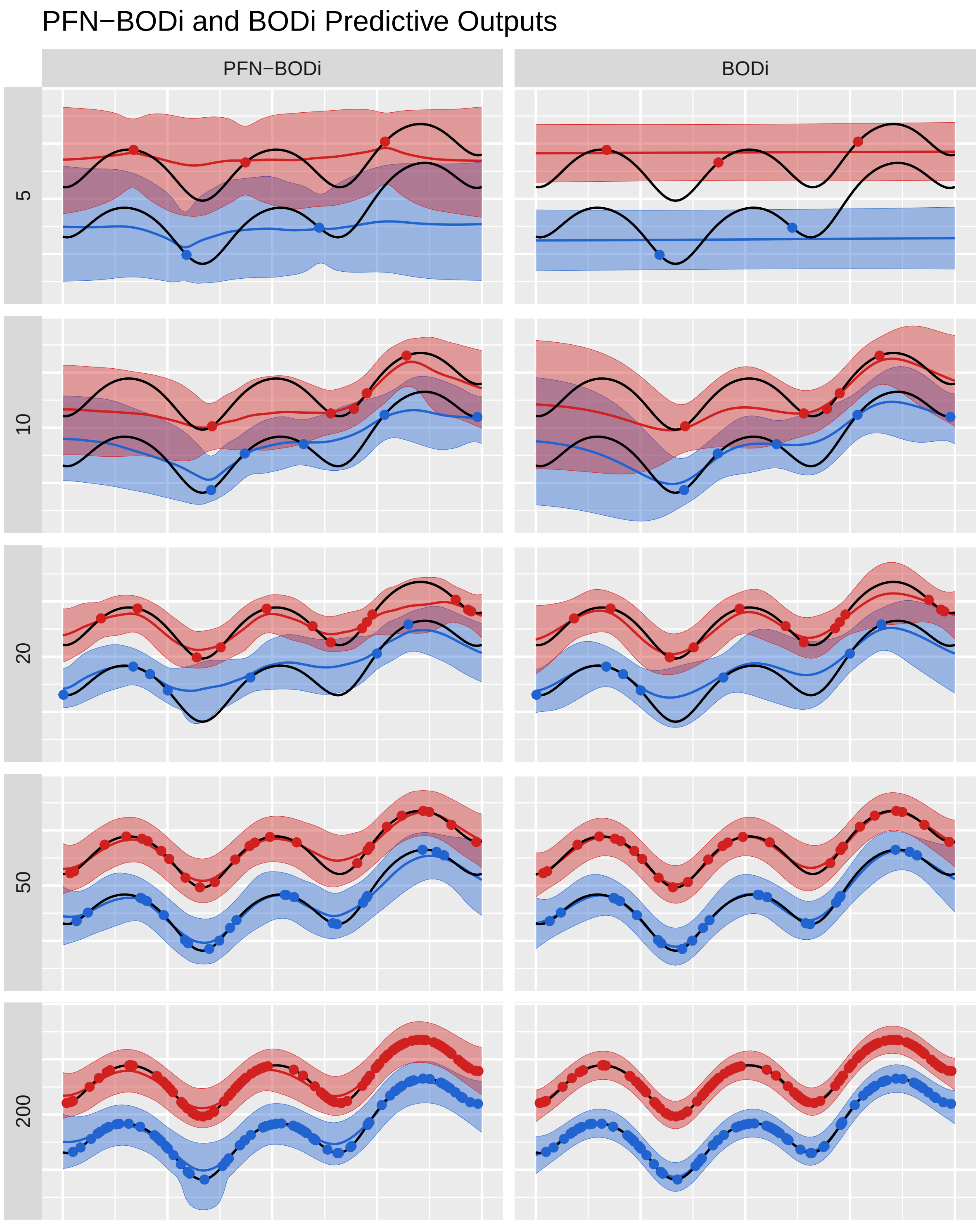

Figure C.1: PFN-BODi and BODi's surrogate model, fit to different amounts of training data. The colored lines represent the model's posterior means for each category, and the region between the region between the 0.025 and 0.975 percentile is shaded. The black lines represent the ground truth of the objective function, and the points illustrate the training data that was drawn from the objective function.

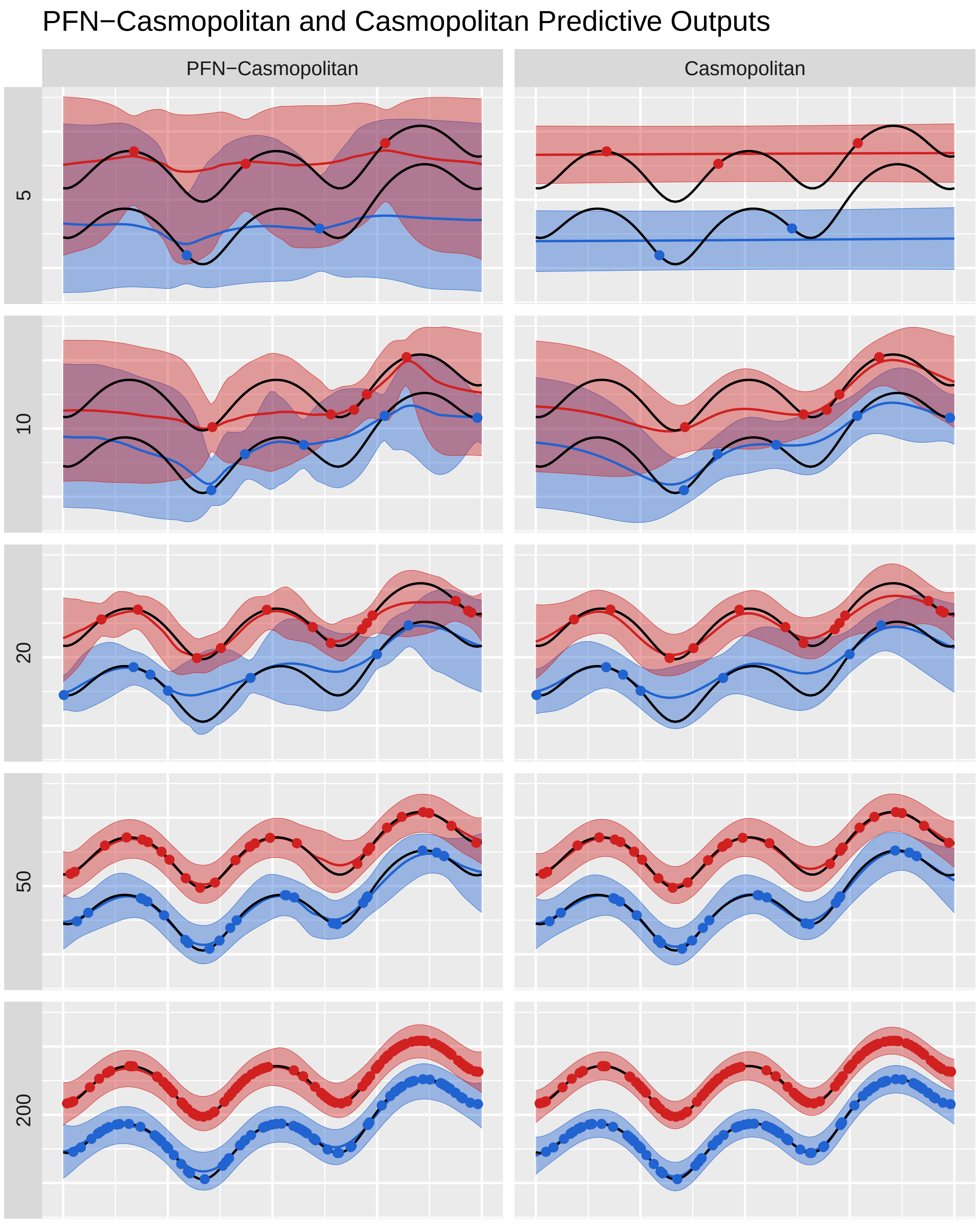

Figure C.2: PFN-Casmopolitan and Casmopolitan's surrogate model, fit to different amounts of training data. The colored lines represent the model's posterior means for each category, and the region between the region between the 0.025 and 0.975 percentile is shaded. The black lines represent the ground truth of the objective function, and the points illustrate the training data that was drawn from the objective function.

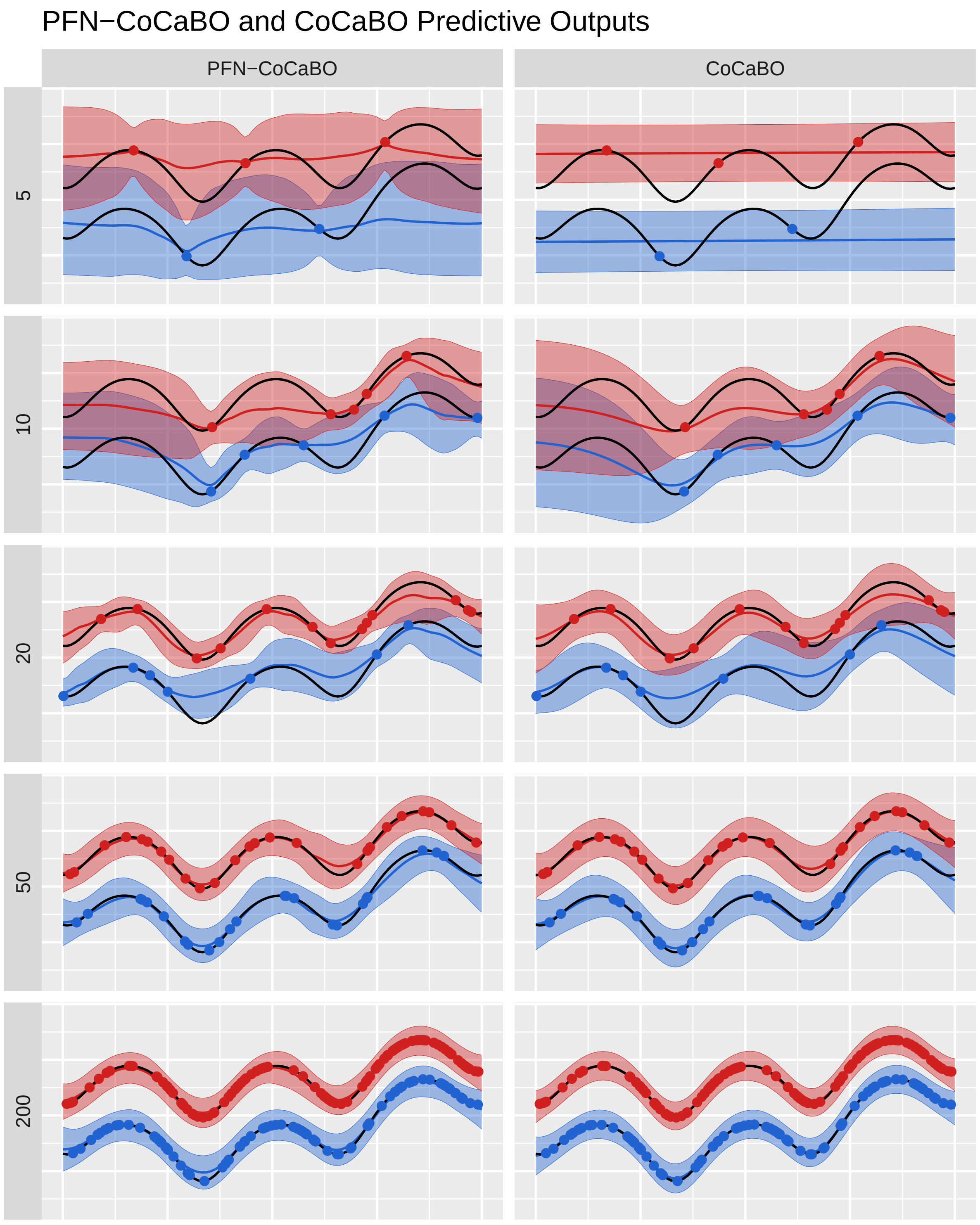

Figure C.3: PFN-CoCaBO and CoCaBO's surrogate model, fit to different amounts of training data. The colored lines represent the model's posterior means for each category, and the region between the region between the 0.025 and 0.975 percentile is shaded. The black lines represent the ground truth of the objective function, and the points illustrate the training data that was drawn from the objective function.

# Appendix D

# Task-Specific BO Experiment Results

In this section, we analyze the performance of different BO methods across the different objective functions that were included in our experiments in section 5.2. Figure D.1 illustrates the trajectories of each optimizer for each task, averaged over all setups.

Figure D.2 depicts the average rank of each optimizer, for each task and iteration. Very loosely, we see that each PFN and its associated GP-based method tend to score similar rankings across the six tasks. We remark that, for the XGBoost optimization task, the CoCaBO-trained PFN performed best out of all models, while it ranked almost worst in other tasks. This highlights the extent to which optimizer performance is dependent on the objective function. With the Rosenbrock function, CoCaBO performed worst out of all the methods tested, including the random baseline. The reasons for this are unclear. We tested the CoCaBO-trained PFN with the same acquisition function and acquisition optimizer as CoCaBO, and it did not perform as poorly. Thus, the reason for CoCaBO's poor performance must have something to do with its surrogate function, but aside from that the reasons are unclear.

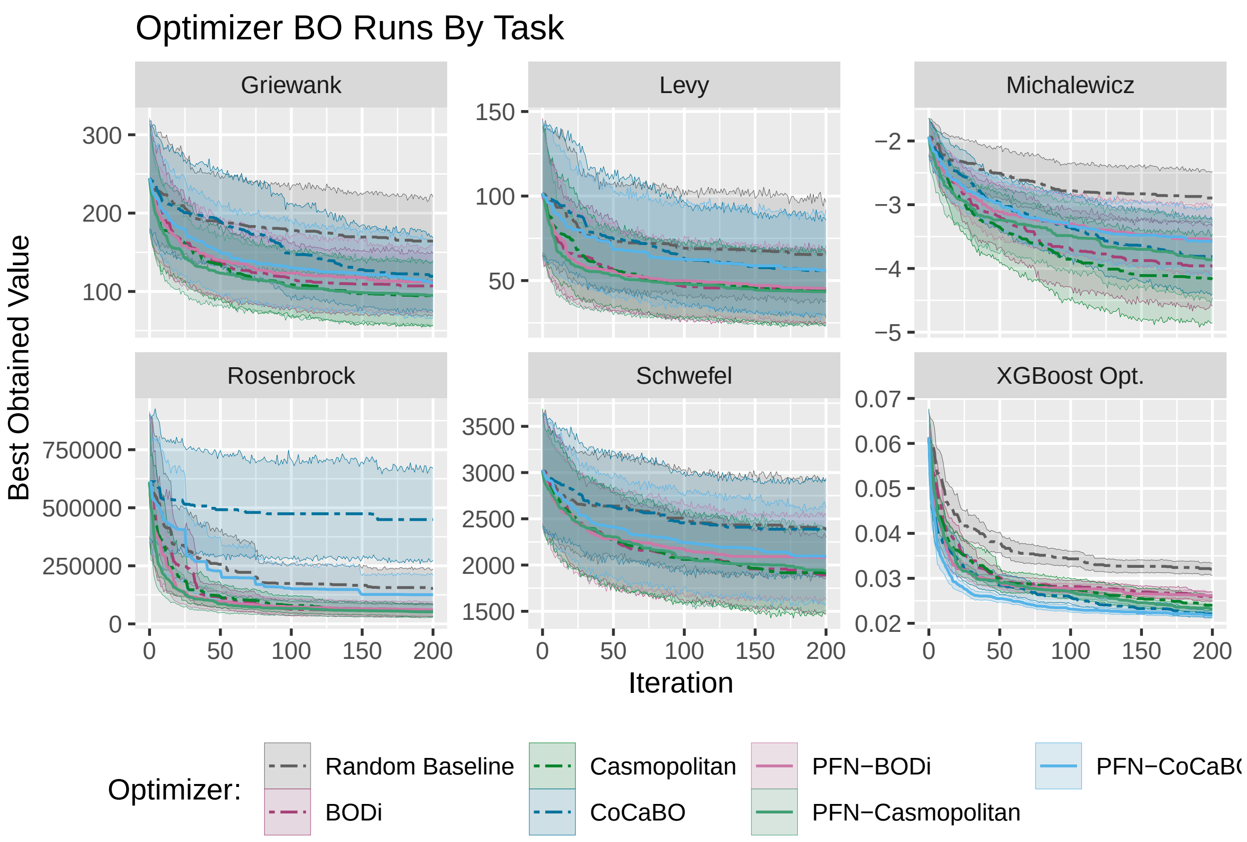

Figure D.1: Trajectories of different optimizers with respect to iteration in BO run, averaged across all setups. 95% bootstrapped confidence intervals are represented by the shaded regions.

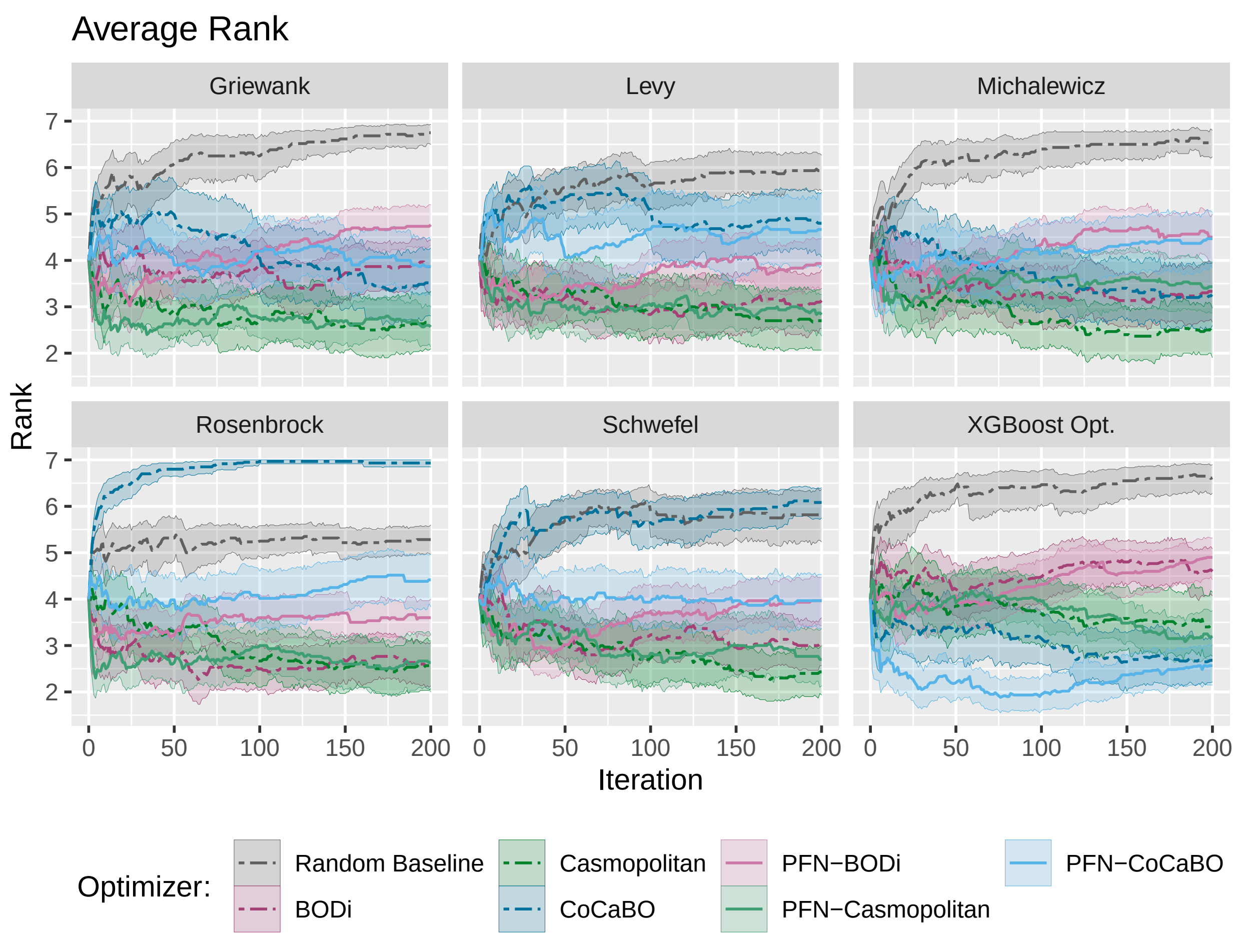

Figure D.2: Ranks of different optimizers with respect to iteration in BO run, averaged across all tasks and all runs. 95% bootstrapped confidence intervals are represented by the shaded regions.

# References

Cowen-Rivers, A. I., Lyu, W., Wang, Z., Tutunov, R., Hao, J., Wang, J., & Bou-Ammar, H. (2020). HEBO: Heteroscedastic evolutionary bayesian optimisation. *CoRR*, *abs/2012.03826*. Retrieved from `https://arxiv.org/abs/2012.03826`

Deshwal, A., Ament, S., Balandat, M., Bakshy, E., Doppa, J. R., & Eriksson, D. (2023). Bayesian optimization over high-dimensional combinatorial spaces via dictionary-based embeddings. *CoRR*, *abs/2303.01774*. http://doi.org/10.48550/ARXIV.2303.01774

Dreczkowski, K., Grosnit, A., & Ammar, H. B. (2023). Framework and benchmarks for combinatorial and mixed-variable bayesian optimization.

Garnett, R. (2023). *Bayesian Optimization*. Cambridge University Press.

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

Mockus, J., Tiesis, V., & Zilinskas, A. (1978). The application of Bayesian methods for seeking the extremum. *Towards Global Optimization*, *2*(117-129), 2.

Müller, S., Feurer, M., Hollmann, N., & Hutter, F. (2023). PFNs4BO: In-context learning for bayesian optimization.

Müller, S., Hollmann, N., Arango, S. P., Grabocka, J., & Hutter, F. (2022). Transformers can do bayesian inference. In *International conference on learning representations*. Retrieved from `https://openreview.net/forum?id=KSugKcbNf9`

Rasmussen, C. E., & Williams, C. K. I. (2006). *Gaussian processes for machine learning.* (pp. I–XVIII, 1–248). MIT Press.

Ru, B., Alvi, A. S., Nguyen, V., Osborne, M. A., & Roberts, S. J. (2019). Bayesian optimisation over multiple continuous and categorical inputs.

Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., & De Freitas, N. (2016). Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, *104*, 148–175. http://doi.org/10.1109/JPROC.2015.2494218

Snoek, J., Larochelle, H., & Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. Retrieved from `https://arxiv.org/abs/1206.`

2944

Solnik, B., Golovin, D., Kochanski, G., Karro, J. E., Moitra, S., & Sculley, D. (2017). Bayesian optimization for a better dessert. In *Proceedings of the 2017 NIPS workshop on bayesian optimization.* December 9, 2017, Long Beach, USA.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., … Polosukhin, I. (2017). Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 30). Curran Associates, Inc. Retrieved from https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

Wan, X., Nguyen, V., Ha, H., Ru, B., Lu, C., & Osborne, M. A. (2021). Think global and act local: Bayesian optimisation over high-dimensional categorical and mixed search spaces.

Weitzman, M. S. (1970). *Measures of overlap of income distributions of white and negro families in the united states.* U.S. Bureau of the Census. Retrieved from https://books.google.de/books?id=G07hHZTDZ0kC

# Selbstständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbständig und nur mit den angegebenen Hilfsmitteln angefertigt habe und dass alle Stellen, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen sind, durch Angaben von Quellen als Entlehnung kenntlich gemacht worden sind. Diese Masterarbeit wurde in gleicher oder ähnlicher Form in keinem anderen Studiengang als Prüfungsleistung vorgelegt.

| | |
|---|---|
| Tübingen, den March 19, 2024 | |
| Ort und Datum | Unterschrift |