Bachelor Programme Datascience and Artificial Intelligence

Enhancing Principal Component Analysis-assisted Bayesian Optimization through Local Embeddings

Adela Gregáňová

Supervisors:
Dr. Elena Raponi & Ivan Olarte Rodriguez

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

01/06/2025

Abstract

Bayesian Optimization (BO) is a technique used for optimizing expensive black-box functions, using a surrogate model constructed from prior evaluations and guided by an acquisition function. To improve BO's scalability to high-dimensional spaces, Principal Component Analysis (PCA) can be used to project the search space onto a lower-dimensional subspace that retains the most relevant information. However, PCA-assisted BO (PCA-BO) can suffer from intrinsic biases that impair convergence, especially in multimodal landscapes with weak global structure. This thesis investigates whether incorporating trust regions and localized PCA strategies can reduce these biases and enhance BO performance. We propose Localized PCA-assisted Bayesian Optimization (LPCA-BO), an algorithm that apart of dimensionality reduction, dynamically adapts the search to different regions of the search space. In LPCA-BO, trust region defines a localized manifold around the current best solution, with PCA applied only to points within these regions to capture locally relevant variance. We evaluate LPCA-BO using the COCO BBOB benchmarking suite, comparing its performance to PCA-BO. Results indicate that LPCA-BO improves both convergence rate and CPU efficiency. By focusing the search in locally informative subspaces and reducing the impact of globally misleading variance, LPCA-BO avoids unnecessary evaluations and constructs more accurate surrogate models, especially when non-uniform variance across parameters is present. LPCA-BO shows significantly improved convergence rates on unimodal functions and performs comparably to state-of-the-art PCA-BO on multimodal functions with strong global structure. Moreover, the algorithm shows significant improvement on multimodal functions with weak global structure, except for function F24. In terms of CPU time, the trust region approach is considerably faster than PCA-BO. These results highlight LPCA-BO's potential as a scalable and robust BO strategy for complex optimization tasks.

Contents

Acknowledgments

I would like to express my deep and sincere gratitude to my thesis supervisor Dr. E. Raponi, for giving me the opportunity to explore this topic for my thesis, providing guidance throughout the course of the semester. In addition, I would like to extend my thanks to my second supervisor I. Olarte Rodriguez for their support and input throughout this journey. I am also sincerely grateful to my friends for their support during challenging times and for making this process more enjoyable. Special thanks goes to Ivan and Luis, who were willing to listen and engage in thoughtful discussions about my thesis topic, helping me stay motivated and go deeper in the topic. I would also like to thank every thesis group members who were there listening to my presentations and asking questions thanks to which I could identify gaps in my knowledge.

1 Introduction

Bayesian Optimization (BO) is a versatile global optimization method suitable for settings where black-box function evaluations are expensive and gradient information is unavailable, making derivative-based algorithms such as gradient descent or Newton's method inapplicable. A common example is hyperparameter optimization for deep neural networks, where the influence of individual parameters or their combinations on model performance is unknown, and evaluation requires fully training and validating the network. By sampling efficiently in the search space, BO is able to minimize the number of costly function evaluations needed to find global optima.

In BO we typically assume an unknown but equal variance as the prior assumption of the model, meaning that each parameter's effect on the objective function is initially treated as equally uncertain, without assuming prior knowledge about parameter relevance. Additionally, the noise in function evaluations is assumed to be homoscedastic, meaning the variance is assumed to be constant across the search space. In this work, we assume noise-free setting.

Rather than exhaustively sampling the search space, Bayesian Optimization performs optimization over a feasible set $A \subseteq \mathbb{R}^d$ by fitting a probabilistic surrogate model, typically a Gaussian Process (GP) regressor, to the available data and then uses an acquisition function to strategically select the next evaluation point. Since the function is expensive to query, it is necessary to sample efficiently and avoid wasting function evaluations on less informative regions. The acquisition function balances exploring regions with high uncertainty to learn more about the function behavior in unexplored regions and exploiting points near known promising areas to refine the search for the global optimum.

The aim of this research is to investigate how to improve the robustness and scalability of PCA-assisted Bayesian Optimization (PCA-BO) in high-dimensional and complex search spaces. Our central research question is: *How can trust regions and localized dimensionality reduction improve the CPU efficiency and convergence reliability of PCA-BO, particularly in settings with non-uniform variance across parameters or where the shape of optimal regions varies strongly across directions (anisotropic basins of attraction)?* Traditional PCA-BO methods can suffer from intrinsic biases introduced by global linear projections, which may fail to capture the local structure of the search space, especially in the presence of multiple local optima or highly non-linear interactions, from which the traditional PCA-BO cannot escape. To address these limitations, we propose augmenting PCA-BO with two key components. First, a trust region mechanism that dynamically adjusts the search area based on optimization progress. Second, a localized PCA strategy that recomputes the projection matrix using samples filtered from within the trust region. Together, these extensions aim to enhance PCA-BO's adaptability and resilience to poor global structure, reducing the likelihood of convergence stagnation. In addition, we hypothesize that this approach can lead to improved computational efficiency in terms of CPU time.

In summary, this thesis investigates whether and how the limitations of PCA-BO can be overcome by incorporating insights from TuRBO's trust region methodology, potentially combining the benefits of both approaches while minimizing their individual drawbacks.

1.1 Thesis Overview

In this bachelor thesis, conducted at the Leiden Institute of Advanced Computer Science (LIACS) under the supervision of Dr. Elena Raponi and Ivan Olarte Rodriguez, we investigate whether incorporating trust regions and localized PCA strategies can enhance BO performance in complex high dimensional spaces. Chapter 1 provides a short general introduction and an overview of the thesis. We begin by reviewing relevant background in Chapter 2, including general Bayesian Optimization (BO), PCA-based dimensionality reduction, PCA-assisted BO, and Trust Region Bayesian Optimization (TuRBO). Chapter 3 presents a detailed description of our proposed trust-region-based Local PCA-BO (LPCA-BO) algorithm, along with the experimental setup. Chapter 4 reports the empirical results, while Chapter 5 interprets the findings, discusses their implications, and suggests directions for future research. Finally, Chapter 6 provides concluding remarks.

2 Related work

Bayesian Optimization (BO) is a widely used sample-efficient strategy for optimizing expensive black-box functions with many tunable parameters. This work investigates whether the use of Principal Component Analysis (PCA) [F.R01] as a dimensionality reduction strategy, when combined with Trust Region Bayesian Optimization (TuRBO) [EPG+20], enhances the effectiveness of BO in high-dimensional spaces. While PCA-assisted BO (PCA-BO) [RWB+20] helps to address some limitations of standard BO, it introduces inherent biases that may compromise optimization performance. TuRBO, on the other hand, addresses the challenges posed by objective functions lacking global structure by partitioning the search space into adaptive trust regions, within which optimization is localized.

2.1 Bayesian Optimization

Bayesian Optimization (BO) is a probabilistic, derivative-free optimization framework designed for expensive-to-evaluate black-box functions. The target function f is assumed to be at least continuous but lacks exploitable structural properties such as convexity or linearity. Given the high computational cost of function evaluations, the optimization process must strategically select evaluation points to maximize information gain. The primary objective is to locate the global optimum $\mathbf{x}^* = \max_{\mathbf{x} \in A} f(\mathbf{x})$ within a predetermined evaluation budget B, where $A \subseteq \mathbb{R}^d$ represents the feasible search domain. This set is usually defined as a hyper-rectangle $x \in \mathbb{R}^d : a_i \leq x_i \leq b_i$ or a d-dimensional simplex $x \in \mathbb{R}^d : \sum_i x_i = 1$ [Fra18]. The function evaluations $f(x)$ might be noisy, in that case assumed independent and normally distributed, with common but unknown variance. In practice, Bayesian Optimization works best and is therefore used for real life applications for problems in lower-dimensional spaces usually up to 20 dimensions.

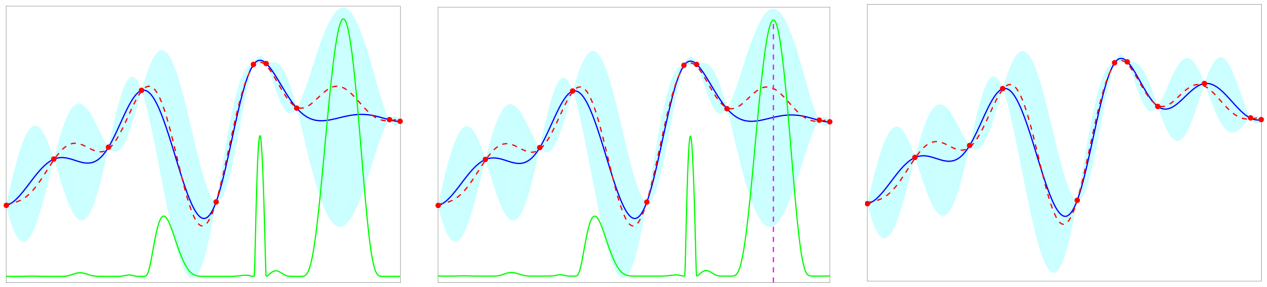
It effectively handles stochastic noise in function evaluations by creating a surrogate model for the objective function and quantifying the uncertainty in that surrogate using Gaussian process regression. This surrogate model is then used to derive an acquisition function that determines the next sampling points. To do this, we use a Gaussian Process Regressor as a surrogate model. Apart of providing us predictions of the function values, it also gives us a measure of uncertainty, how confident the model is in different regions of the space. Then, we use this model to define an acquisition function, which tells us where to sample next, taking into account exploring uncertain areas and exploiting promising ones.

The general Bayesian Optimization algorithm is formalized in Algorithm 1. In summary, the algorithm begins by collecting an initial dataset of n_0 function evaluations using a space-filling experimental design and placing a Gaussian process prior on the unknown function f. Subsequently, while the budget is not exhausted, the algorithm updates the posterior distribution over f using all available data (the surrogate model), optimizes the acquisition function to determine the next evaluation point x, sample this point, evaluate $f(x)$ and and incorporates this new observation into the dataset.

Figure 1 shows three example iterations of the Bayesian optimization procedure. Each plot displays the posterior mean (dark blue) and confidence intervals (light blue) estimated by a Gaussian Process model. The plot also shows the acquisition function (green). The acquisition is high where the model predicts a high objective function value (exploitation) and where the prediction uncertainty is high (exploration). As we can see, the chosen point to sample (vertical red dotted line) has strong potential to improve upon the current best observation. In contrast, other regions with similarly high uncertainty are not selected, as they are correctly predicted to offer little improvement. [SSW+16]

Algorithm 1 Pseudo-code for Bayesian optimization [Fra18]

1: Place a Gaussian process prior on f
2: Observe f at n_0 points according to an initial space-filling experimental design. Set $n = n_0$.
3: **while** $n \leq N$ **do**
4: Update the posterior probability distribution on f using all available data
5: Let x_n be a maximizer of the acquisition function over x, where the acquisition function is computed using the current posterior distribution.
6: Observe $y_n = f(x_n)$.
7: Increment n
8: **end while**
9: **return** a solution: either the point evaluated with the largest $f(x)$, or the point with the largest posterior mean.



(a) (b) (c)

Figure 1: Illustration of Bayesian Optimization over three iterations. Red points indicate evaluated locations, and the red dotted line shows the true (unknown in practice) objective function. The blue curve represents the GP posterior mean, with shaded regions denoting ±2 standard deviations. The green curve is the acquisition function. [Rai22]

2.1.1 Initial Design of Experiments (DoE)

To provide the surrogate model with informative initial data that captures the underlying function's behavior across the design space, it is essential to strategically select and distribute initial sample points before the iterative acquisition process begins. Common design of experiments (DoE) techniques used for this purpose include Latin hypercube sampling [FLS05] and Sobol sequences [Sob67], which promote good coverage of the input space.

The initial set of points for Bayesian Optimization used in this thesis is generated and subsequently evaluated using an Optimal Latin Hypercube Sampling (OLHS) [FLS05] strategy, performed in the original high-dimensional search space as part of the Design of Experiments (DoE).

OLHS, derived from stratified sampling, ensures that each input variable's range (each dimension of the parameter space) is evenly partitioned into equally probable strata [FLS05]. A single sample is then drawn from each stratum, resulting in a well-distributed set of initial points that promotes broad and uniform exploration of the parameter space capturing the general structure of the objective function early on.

This sampling strategy is particularly effective in cases where the objective function is governed primarily by main effects and low-order interactions. Under such conditions, LHS significantly reduces the variance of the estimates by filtering out variance from additive noise components. As a result, LHS is widely used in surrogate modeling frameworks such as Gaussian Processes in Bayesian Optimization, where efficient use of limited function evaluations is critical due to a budget constraint.

The points sampled are denoted as $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_{n_0}]^\top \in S^{n_0}$ and its corresponding function values as $\mathbf{y} = (f(\mathbf{x}_1), \ldots, f(\mathbf{x}_{n_0}))^\top$, $n \leftarrow n_0$.

2.1.2 Gaussian Process Regression Model

In Bayesian Optimization, we place a prior distribution over the objective function we wish to optimize, and use a surrogate model to infer a posterior predictive distribution over the objective values at all feasible points in the search space. A commonly used surrogate model is the Gaussian Process (GP) regressor, which provides a way to model uncertainty in the function being optimized.

A Gaussian Process (GP) is a type of stochastic process that enables us to build a probabilistic model of an unknown function by incorporating prior knowledge. In regression tasks such as fitting a function to the available data, GPs offer an elegant solution to the problem of fitting a function to observed data. Given a set of known function values at specific input points, there are potentially infinitely many functions that fit this data. A GP addresses this uncertainty by assuming that the function values at any finite collection of input points, $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \ldots, \mathbf{x}_k \in \mathbb{R}^d$, follow a joint Gaussian distribution, meaning each individual value is normally distributed, and any subset of values has a multivariate normal distribution. The collection of the input points can be expressed together as the unknown vector

$$\mathbf{f} = \big[f(\mathbf{x}_1), f(\mathbf{x}_2), \ldots, f(\mathbf{x}_k)\big] \in \mathbb{R}^k,$$

where each input $\mathbf{x}_i \in \mathbb{R}^d$ is a d-dimensional vector.

The multivariate Gaussian distribution is characterized by a mean vector μ and a covariance matrix Σ:

$$\mathbf{f} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma).$$

The mean vector describes the expected values of the function constructed by evaluating a mean function μ_0 at each x_i, and the covariance matrix captures the variability and correlation between values at different input locations [GKD19]. Instead of fitting a fixed function, GP regression makes predictions at new (test) inputs by conditioning the joint Gaussian distribution of the training and test points. The outcome is a predictive distribution for each test point, providing both a mean prediction (the most probable function value) and a measure of uncertainty (the variance). This is a key advantage of GPs, they do not just make point predictions but quantify uncertainty.

Mathematically, the joint distribution over the training outputs $\mathbf{Y}$ and the test outputs $\mathbf{T}$ is modeled as a single multivariate Gaussian. The conditional distribution $P(\mathbf{T} \mid \mathbf{Y})$ (in our case trained on Z_r, y), which is central to prediction, is also Gaussian due to the closure property of the multivariate normal distribution under conditioning [GKD19].

The covariance matrix Σ is defined using a kernel function $k(x, x')$, which encodes prior assumptions about the properties of the underlying function, such as smoothness, periodicity, or stationarity. The kernel quantifies the similarity between any two inputs x_i and x_j; when inputs are similar, their corresponding outputs $f(x_i)$ and $f(x_j)$ are expected to be similar as well. In this way, the kernel determines how information from the training data generalizes to new, unseen points. This behavior is governed by kernel hyperparameters, such as the length scale, which controls how strongly function values at nearby inputs influence each other, or the smoothness parameter α in the power exponential kernel, which controls how rapidly the function can change.

Figure 2 shows random functions sampled from a Gaussian process prior with a power exponential kernel, for different values of the smoothness parameter α (decreasing from left to right). Lower values of α result in less smooth, more rapidly changing functions, reflecting different prior beliefs about the function's local variability. [Fra18]

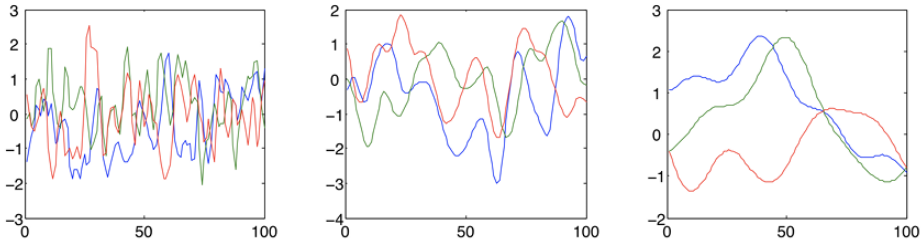


Figure 2: Random functions drawn from a GP prior with a power exponential kernel. Each plot uses a different value of the smoothness parameter α (decreasing left to right), affecting how rapidly $f(x)$ varies.[Fra18]

In practice, we only need to define a distribution over the function values at a finite set of

inputs $x_1, \ldots, x_N$. Given some observed input-output pairs $(x, f(x))$ and new input locations x^* where we wish to make predictions, we assume that the joint distribution over the training outputs f and the test outputs f^* is Gaussian. From this joint distribution, we can derive the conditional distribution:

$$f^* \mid f \sim \mathcal{N}(\mu^*, \Sigma^*),$$

where μ^* and Σ^* are the posterior predicted mean and covariance, respectively. This provides a full probabilistic description of the function at the test points, including both the predicted values and the associated uncertainty. This relationship generalizes to the multivariate setting in Gaussian Processes, allowing for sampling from predictive distributions and quantifying uncertainty.

Typically, the objective function is modeled using a Gaussian process prior with zero mean:

$$f \sim \mathcal{GP}(0, k(\cdot, \cdot)),$$

where $k : \mathcal{S} \times \mathcal{S} \to \mathbb{R}$ is a positive definite function (kernel) that computes the autocovariance of the process. When a Gaussian likelihood is assumed, the resulting posterior remains analytically tractable and is itself a Gaussian process:

$$f \mid \mathbf{y} \sim \mathcal{GP}(\hat{f}(\cdot), k'(\cdot, \cdot)),$$

where $\hat{f}$ and k' denote the posterior mean and covariance functions, respectively. At a test point $\mathbf{x}$, the mean $\hat{f}(\mathbf{x})$ provides the maximum a posterior (MAP) estimate of $f(\mathbf{x})$, while the variance $\hat{s}^2(\mathbf{x}) := k'(\mathbf{x}, \mathbf{x})$ captures the uncertainty of this prediction. This posterior defines the Gaussian Process Regression (GPR) model. [RWB+20]

Finally, training a GP regression model involves optimizing the kernel hyperparameters, such as length scale and variance, to maximize the marginal likelihood of the observed data, fitting the prior assumptions to the structure of the actual observations.

2.1.3 Acquisition Functions

Acquisition function is a heuristic used in BO that guides the selection of the next points for evaluation, helping to balance the trade-off between exploring new regions of search space and exploiting regions with high predicted values. It uses the posterior distribution generated from the surrogate model (predicted mean and variance of the surrogate model prediction) to determine where to sample next, in order to use the available budget effectively and find the global optimum of the objective function.

There are many other types of acquisition functions, each with its own way of heuristics about improvement or uncertainty. One of the most commonly used acquisition functions is Expected Improvement (EI). EI assumes sampling is valuable if it improves over the best observed value by quantifies the expected gain, and selects points likely to yield improvement, balancing mean and uncertainty. Probability of Improvement (PI) on the other hand focuses on the probability that $f(x)$ exceeds the current best, regardless of how large the improvement is. Another type of acquisition function is Upper Confidence Bound (UCB), which selects points with the highest combination of mean and uncertainty, weighted by a tunable parameter β. It offers a flexible trade-off

between exploration and exploitation, favoring points with high predicted value and high uncertainty.

Unlike EI, which assumes sampling is valuable if it improves over the best observed value, there are more advanced methods that don't assume improvement is the only goal. These include: Knowledge gradient, Entropy Search and Predictive Entropy Search, Multi-Step Optimal Acquisition Functions. These approaches, on the other hand, aim to maximize information gain (Entropy Search and Predictive Entropy Search) or long-term decision quality (Knowledge Gradient), rather than immediate improvement.

Another popular acquisition strategy is Thompson Sampling, which unlike the other acquisition functions samples a full function from the surrogate model rather than points and selects the next evaluation point by optimizing this sampled function.

When optimizing an acquisition function, the next point to evaluate is chosen by either maximizing or minimizing the acquisition function, depending on the optimization task and the specific type of acquisition function used. The Expected Improvement (EI) function is computationally inexpensive to evaluate and supports efficient computation of both its first- and second-order derivatives. This makes it well-suited for use with continuous optimization methods relying on gradient information. Consequently, implementations of the EI algorithm often involve gradient-based optimizers to efficiently identify the next point to evaluate. Common optimizers include L-BFGS-B [LN89], and SLSQP [MZL22] (Sequential Least Squares Programming), frequently used in constrained optimization settings.

2.1.4 Expected Improvement (EI)

The general intuition behind Expected Improvement (EI) is to identify, among all candidate points, the one that offers the highest potential to surpass the best function value observed so far. This can occur either because the model predicts a high objective value at that point (exploitation), or because there is significant uncertainty in that region (exploration). The improvement is defined as the difference between the function value at the candidate point and the current best value.

As discussed earlier, Bayesian Optimization is particularly suited for optimizing expensive black-box functions where the number of evaluations is limited. In such cases, each evaluation must be strategically chosen to extract the most useful information.

Suppose that, after n evaluations, we have observed input-output pairs

$$\{(x_i, y_i)\}_{i=1}^{n},$$

and let f_n^* denote the best function value observed so far. The challenge is to decide where to evaluate next in order to maximize the benefit of the remaining evaluations.

To formalize this, consider sampling a new point x, and define the improvement as:

$$\text{Improvement}(x) = \max(f(x) - f_n^*, 0) = [f(x) - f_n^*]^+.$$

However, the true value $f(x)$ is unknown before evaluation. Instead, since we model the objective function f with a Gaussian Process (GP), we have a posterior distribution over $f(x)$, characterized by a predictive mean $\mu_n(x)$ and variance $\sigma_n^2(x)$. We therefore treat $f(x)$ as a random variable and compute the *Expected Improvement (EI)*, defined as:

$$\text{Expected Improvement} \quad (\text{EI})_n(x) = \mathbb{E}_n\left[[f(x) - f_n^*]^+\right]$$

where $\mathbb{E}_n[.]$ is taken under the posterior distribution of the GP conditioned on the data $x_{1:n}, y_{1:n}$.

This quantity represents the expected amount by which the function value at point x exceeds the current best observation f_n^*, and is used to guide the next sampling decision. The next evaluation point is then selected by maximizing the expected improvement.

This choice balances exploitation, which is sampling where the predicted mean $\mu_n(x)$ is high, and exploration, sampling where the uncertainty $\sigma_n(x)$ is large, high posterior standard deviation. The expected improvement value is 0, at locations where the objective function has already been evaluated. This is due to the fact that at those points, the posterior standard deviation vanishes, and the posterior mean cannot exceed the best function value observed thus far. Evaluating points with high uncertainty helps reduce model uncertainty and explore unknown regions of the search space. On the other hand, evaluating points with high predicted mean relative to the current best is advantageous, as such points are more likely to contain a good approximation of the global optimum. We define the expected difference in quality between a candidate point x and the best previously observed point as $\Delta_n(x) := \mu_n(x) - f_n^*$.

Figure 3 illustrates how the expected improvement $\text{EI}_n(x)$ increases with both $\mu_n(x)$ and $\sigma_n(x)$. Contours of equal expected improvement in the $(\mu_n(x), \sigma_n(x))$-plane show how EI balances exploration and exploitation: higher $\mu_n(x)$ targets promising regions, while higher $\sigma_n(x)$ encourages evaluations in uncertain areas.

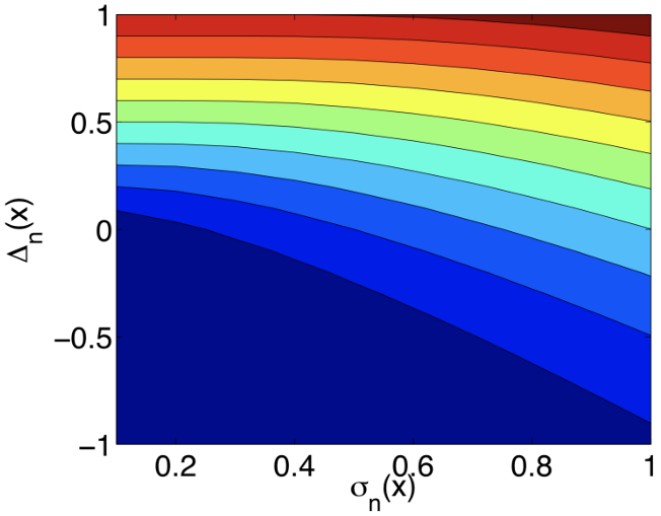


Figure 3: The contours of $EI_n(x)$ in terms of the expected difference in quality between the proposed point and the best previously evaluated point and the posterior standard deviation. [Fra18]

2.1.5 Limitation of BO

Although Bayesian Optimization (BO) is a popular framework for black-box function optimization problems, it is most effective in input spaces up to around 20 dimensions. As the dimensionality of the input space increases, BO becomes less effective due to several inherent challenges. The most notable is the curse of dimensionality, where the volume of the search space grows exponentially with the number of dimensions. This growth makes effective exploration more expensive and increases the risk of encountering many local optima, while the global optimum becomes harder to identify. Furthermore, many high-dimensional objective functions exhibit heterogeneous behavior, meaning that the function's properties can vary significantly across different regions of the domain. Standard BO typically uses a global surrogate model, such as a Gaussian Process (GP), which assume stationary properties such as constant variance throughout the space. This assumption often fails in practice, leading to poor predictive performance in complex, high-dimensional landscapes.

2.2 Principal Component Analysis assisted Bayesian Optimization (PCA-BO) algorithm

One approach to address these issues is PCA-BO algorithm [RWB+20], which combines BO with Principal Component Analysis (PCA). The purpose of this method is to identify a low-dimensional linear subspace that captures the directions of greatest variance in the data. Formally, the objective function is defined as $f : S \subset \mathbb{R}^D \to \mathbb{R}$, where $S = [x_{\min}, x_{\max}]$ represents the domain with box constraints. PCA-BO projects the original D-dimensional space onto a reduced r-dimensional subspace ($r < D$), and BO is conducted within this lower-dimensional space. This dimensionality reduction makes surrogate modeling and acquisition function optimization more efficient and scalable.

In order to scale Bayesian Optimization (BO) to higher-dimensional and more complex parameter spaces, we apply Principal Component Analysis (PCA) [F.R01] for dimensionality reduction. PCA iteratively performs an orthogonal linear transformation of the original search space, projecting it into a new feature space where the data become more separable. The features are selected to retain the maximum possible variance from the original space. The advantage of this transformation is that it enables the most computationally intensive steps of BO, Gaussian Process model fitting and acquisition function optimization to be carried out in a lower-dimensional space, significantly reducing the computational resources needed to find the next promising point to sample next. The selected point is then mapped back to the original space, evaluated, compared to the current optimum, and added to the set of evaluated points along with its corresponding function value.

Algorithm 2 presents the PCA-assisted Bayesian Optimization procedure. The algorithm begins by generating an initial DoE of n_0 points using optimal Latin hypercube sampling in the original D-dimensional space $\mathcal{S}$ and evaluating the objective function f at these locations. The core optimization loop incorporates several key steps that distinguish PCA-BO from standard BO: first, the algorithm centers the data by subtracting the mean $\boldsymbol{\mu}$ and applies rescaling through matrix $\mathbf{W}$ to normalize the feature space (lines 5-6). Next, PCA is performed on the preprocessed data to obtain the principal components $\mathbf{P}_r$ and the reduced-dimensional representation $\mathbf{Z}_r \in \mathbb{R}^r$ where $r \ll D$ (lines 7-8). The Gaussian Process Regression model is then trained on this lower-dimensional

representation rather than the original space (line 9). The acquisition function is optimized in the reduced space $\mathcal{S}'$ to find the next query point $\mathbf{z}'$ (line 10), which is subsequently mapped back to the original D-dimensional space using the inverse PCA transformation (line 11). Finally, the objective function is evaluated at this reconstructed point, and the dataset is augmented with the new observation (lines 12-13). This process continues until the stopping criterion is met.

PCA-BO can significantly reduce computational cost on high-dimensional problems while maintaining competitive convergence rates on functions with a strong global structure. On multimodal benchmark problems where such structure is present, the algorithm achieves state-of-the-art Bayesian Optimization performance. However, its effectiveness degrades on multimodal problems with weak or fragmented global structure.

In terms of computational efficiency based on the CPU time required to complete each optimization run, except for lower-dimensional problems, where standard BO remains efficient, PCA-BO consistently reduces average CPU time. This improvement arises from conducting the most computationally intensive steps, namely, training the Gaussian Process model and optimizing the acquisition function in a lower-dimensional search space, which is significantly more efficient than performing them in the full-dimensional domain.

Algorithm 2 PCA-assisted Bayesian Optimization [RWB+20]

1: **procedure** PCA-BO(f, $\mathcal{A}$, $\mathcal{S}$) ▷ f: objective function, $\mathcal{A}$: infill-criterion, $\mathcal{S}$: search space, $\mathcal{S}'$: lower-dimensional search space
2: Create $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{n_0}]^\top \subseteq \mathcal{S}^{n_0}$ with optimal Latin hypercube sampling
3: $\mathbf{y} \leftarrow (f(\mathbf{x}_1), f(\mathbf{x}_2), \ldots, f(\mathbf{x}_{n_0}))^\top$, $n \leftarrow n_0$
4: **while** the stop criteria are not fulfilled **do**
5: Centering: $\tilde{\mathbf{X}} \leftarrow \mathbf{X} - \mathbf{1}_n \boldsymbol{\mu}$
6: Rescaling: $\mathbf{X}' \leftarrow \tilde{\mathbf{X}}\mathbf{W}$
7: $\boldsymbol{\mu}', \mathbf{P}_r \leftarrow \mathrm{PCA}(\mathbf{X}')$
8: Mapping to $\mathbb{R}^r$: $\mathbf{Z}_r \leftarrow \mathbf{P}_r(\mathbf{X} - \mathbf{1}_n \boldsymbol{\mu}')^\top$
9: GPR training: $\hat{f}, s^2 \leftarrow \mathrm{GPR}(\mathbf{Z}_r, \mathbf{y})$
10: $\mathbf{z}' \leftarrow \arg\max_{\mathbf{z} \in \mathcal{S}'} \mathcal{A}(\mathbf{z}; \hat{f}, s^2)$
11: Mapping to $\mathbb{R}^D$: $\mathbf{x}' \leftarrow \mathbf{P}_r^\top \mathbf{z}' + \boldsymbol{\mu}' + \boldsymbol{\mu}$
12: $y' \leftarrow f(\mathbf{x}')$
13: $\mathbf{X} \leftarrow \mathbf{X} \cup \{\mathbf{x}'\}$, $\mathbf{y} \leftarrow (\mathbf{y}^\top, y')^\top$, $n \leftarrow n + 1$
14: **end while**
15: **end procedure**

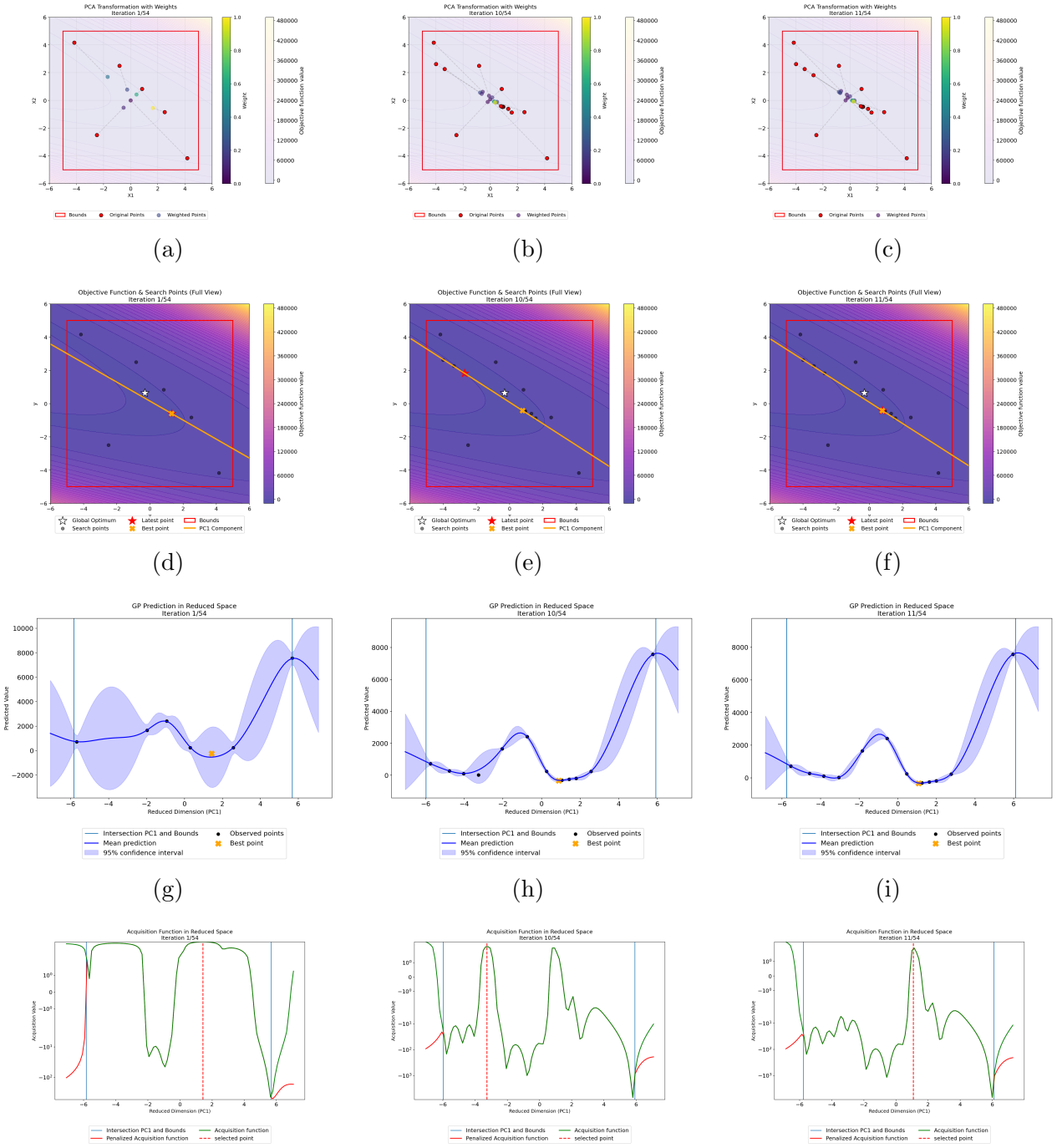


Figure 4: Visualization of the 1st, 21st, and 54th iteration of the PCA-BO algorithm applied to a low or moderate conditioning Rosenbrock function (BBOB suite F8). Each column corresponds to one iteration. Rows represent: (a-c) the weighting of points used for fitting the PCA line, (d-f) the true objective function landscape with sampled points, (g-i) the fitted Gaussian Process (GP) models, and (j-l) the corresponding acquisition functions (Penalized LogEI). The weights indicate the relative influence of points in computing the principal component direction at each iteration. A complete example of this run, along with additional examples illustrating the behavior of the algorithm on various BBOB functions, is available in the accompanying Github repository.

2.2.1 Dimensionality Reduction via Principal Component Analysis

Principal Component Analysis (PCA) is an unsupervised dimensionality reduction technique that projects high-dimensional data onto a lower-dimensional subspace, capturing the most informative aspects of the data. After weighting our data points, PCA is applied to identify a linear transformation from the original space to a lower-dimensional latent space by selecting the top $d' << d$ principal components. This is particularly valuable in high-dimensional settings, as it helps limit the curse of dimensionality by discarding less informative directions while preserving the maximum variance possible. In summary, PCA learns a linear transformation from the points evaluated so far and selects only the dimensions that capture most of the variance.

The procedure involves computing the sample mean μ ($\mu = \frac{1}{n}\sum_{i=1}^{n} X'_{i1}, ..., \frac{1}{n}\sum_{i=1}^{n} X'_{id}$) of data matrix X' and centering the data by subtracting the mean of each variable to obtain the mean-centered matrix $X'' = X' - 1_n\mu'$. Here 1_n is an n-dimensional vector of ones. Subtracting the mean centers the data around the origin, which is essential for PCA to correctly identify the directions of maximum variance (i.e., the principal components). Now that we centered the data around the origin, we can compute the covariance matrix C ($C = \frac{1}{n-1}X''^{\top}X''$) which captures the pairwise relationships between variables. Positive covariance indicates that variables increase or decrease together. The next step involves calculating the eigenvalues and eigenvectors of the covariance matrix by performing eigendecomposition of the covariance matrix $C = PDP^{-1}$. The column vectors of P are the eigenvectors (principal components) that define directions of maximum variance. On the other hand, the corresponding eigenvalues are contained within the diagonal matrix $D = diag(\sigma_1^2), ..., \sigma_d^2$. Each eigenvalue σ_i^2 represents the variance of the data along the principal component p_i (the i-th column of P), i.e., $\sigma_i^2 = \text{var}(X''p_i)$. The eigenvectors within the orthogonal matrix P rotate the space to transform the standard basis of $\mathbb{R}^D$ into a new basis that aligns with the directions of maximum variance. To reduce dimensionality, we sort the principal components by their variances (eigenvalues) in descending order and select the top r components. The value of r is chosen such that the cumulative variance explained by these r components is at least a predefined threshold α; we use $\alpha = 95\%$ in our experiments. Based on these most significant components, we reduce dimensionality while retaining most of the original data's structure.

Finally, the data X along these principal components is then projected onto this new r-dimensional subspace Z_r by first centering the matrix $\overline{X}$ (which already has subtracted mean) via subtraction of the mean matrix product $1_n\mu'$ (which is the mean from PCA), and then multiplying the result (transposed) with the eigenvector matrix P_r:

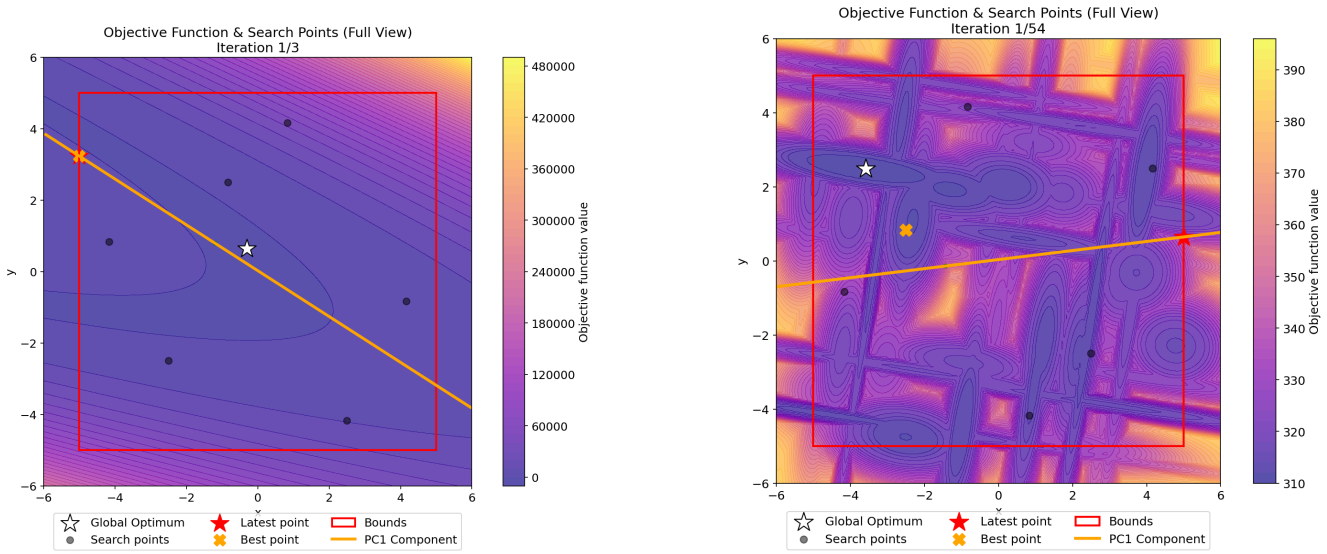
$$Z_r = P_r(\overline{X} - \mathbf{1}_n\mu')^{\top}$$

In this PCA-based dimensionality reduction approach for the dimensionality reduction we do not work with the scaled data, rather the original function values since scaling $\overline{X}$ with weights W would make the inverse mapping no longer a simple linear transformation. Furthermore, the reason why we subtract the sample mean of the PCA is that it makes the reduced representation more meaningful and structured for optimization since in this way, the principal axes of these contours will be parallel to the principal components after applying P_r.

2.2.2 Intrinsic Biases of PCA-BO

Although PCA-BO improves computational efficiency by reducing the dimensionality of the search space, it is inherently limited by its reliance on a linear projection of the original domain. Because the optimization is restricted to the linear subspace defined by the principal components, the algorithm may overlook promising regions that lie outside of this manifold. This limitation becomes particularly problematic when the objective function exhibits complex nonlinearities or lacks a well-defined global trend. In such cases, PCA-BO may struggle to detect meaningful isocontours within the function landscape and can become trapped in suboptimal regions.

Figure 5 demonstrates the effectiveness of PCA in learning the underlying manifold structure for functions with contrasting optimization landscapes. More specifically a function with multimodal landscape 5b and a function with unimodal landscape 5a. We compare PCA's performance on the unimodal Rosenbrock rotated function (F9) from the bbob test suite against the multimodal Gallagher 101 peaks function (F21), which exhibits weak global structure. For the unimodal function, PCA effectively captures the relevant search direction with the first principal component (orange line) aligning closely with the valley structure that leads toward the global optimum. The search points naturally span the single basin of attraction. In contrast, the multimodal function presents significant challenges for PCA due to its complex landscape of multiple peaks and valleys. The principal components fail to capture meaningful search directions because the irregular distribution of local optima prevents the identification of coherent global structure, resulting in projections that do not effectively guide the optimization process toward the most promising regions.



(a) PCA projection on a unimodal, Rosenbrock rotated (F9) function.

(b) PCA projection on a multimodal, Gallagher 101 peaks (F21) function.

Figure 5: Comparison of PCA-derived manifolds for different function types. For the unimodal function (left), standard PCA captures the relevant search directions because the sampled points span the single basin of attraction. For the multimodal function (right), PCA fails to capture the global structure if no samples are placed within the major basins of attraction, leading to misleading directions.

2.3 TuRBO addressing BO limitation

To address the limitations of global surrogate modeling in high-dimensional settings, Trust Region Bayesian Optimization (TuRBO) algorithm was introduced [EPG+20]. Rather than relying on a single global model, TuRBO maintains several local models, each associated with its own trust region, a hyperrectangle polytope centered around the current best solution. These models operate independently using Gaussian Processes (GPs) and apply Thompson Sampling to propose new candidate points. At each iteration, the algorithm performs Thompson Sampling across all trust regions by drawing posterior samples from each local GP and selecting the most promising candidates based on these samples. Sample allocation among the trust regions is governed by an implicit multi-armed bandit strategy, which favors regions demonstrating recent improvement. This algorithm achieves global optimization through maintaining several independent local models.

Each trust region is initialized with a base side length L_{init}, which is adapted during the optimization process. The side lengths in each dimension are scaled according to the Automatic Relevance Determination (ARD) [ARD] lengthscales inferred by the GPIf the algorithm achieves a series of improvements within the region, the region is expanded. Conversely, consecutive failures result in the region being shrunk. When a trust region becomes too small, it is discarded and re-initialized at a new location in the input space after a new Design of Experiment points.

The general TuRBO algorithm maintains multiple trust regions in parallel, each with an independent GP surrogate model. However, a simplified variant known as TuRBO-1 maintains only a single trust region and is especially relevant to this work due to its efficiency and simplicity. The TuRBO-1 optimization loop begins by sampling n_init points using Latin Hypercube Sampling (LHS) and evaluating the objective function at these locations. A GP model is then fitted to the collected data. The trust region is defined around the current best point and managed using ARD-weighted box constraints. If optimization improves, the trust region is expanded. Otherwise, it shrinks.

2.3.1 Proposed Integration of TuRBO and PCA-BO

Inspired by both PCA-BO [RWB+20] and TuRBO-1 [EPG+20], this research proposes integrating trust-region dynamics into the PCA-BO framework. At each iteration, a trust region is defined around the best observed solution, and PCA is applied locally within this region to identify the most informative directions. A new candidate is then sampled within the resulting low-dimensional subspace. This approach aims to combine the scalability of PCA-based dimensionality reduction with the robustness to noise and advanced reasoning about uncertainty, of a local trust-region-based exploration. By performing dimensionality reduction within a localized region of the search space, the method seeks to overcome PCA-BO's limitations on functions with weak global structure, thus improving performance in complex and challenging optimization landscapes.

3 Methodology

Our approach began with the implementation of the PCA-BO algorithm using BoTorch. We then extended this implementation by developing Local PCA-BO (LPCA-BO) algorithm, which introduces a dynamically updated trust region centered around the best observed point at each iteration. Within this localized region, we perform dimensionality reduction using Principal Component Analysis (PCA), followed by a Gaussian Process (GP) model optimizing the acquisition function in the resulting lower-dimensional subspace. Our implementation of Local PCA-BO algorithm is publicly available in this GitHub repository.

3.1 Local Optimization using Trust Region

In heterogeneous multi-modal objective functions that lack global structure such as those encountered in Reinforcement learning problems with sparse rewards, the function is assumed close to being constant. Traditional Gaussian Process (GP) models assume stationarity, meaning the length-scale and signal variance are treated as constant throughout the domain. This assumption becomes problematic in such settings. Moreover, the curse of dimensionality introduces large regions of the search space with inherently high posterior uncertainty. When combined with myopic acquisition functions, such as Expected Improvement (EI), which do not consider the long-term impact of sampling decisions, this often leads to excessive exploration and insufficient exploitation of promising regions.

Each trust region defines a local model that executes an independent optimization run. PCA is applied to the points within the region, enabling more efficient GP modeling and acquisition in a lower-dimensional manifold. In our implementation, the trust region is defined as a hyperrectangle centered around the best observed solution, with bounds updated based on the optimization progress.

Global optimization is achieved through dynamic adjustment of the trust region's size and position within the search space balancing exploration and exploitation. A smaller trust region encourages local exploitation by focusing the search near the current best solution, while a larger trust region performs more exploratory behavior. To prevent stagnation in local optima, if the trust region shrinks below a predefined minimum size, it is restarted, allowing the algorithm to explore new areas of the search space. This adaptive mechanism guides the search process by allowing the trust region to progressively explore larger search space, increasing the likelihood of escaping local optima and converging to the global optimum, even in complex high-dimensional landscapes.

3.2 Local PCA-BO

The complete pseudocode of the algorithm can be found in Algorithm 3. At the start, lists for tracking sampled points and their corresponding function evaluations are initialized. Those initial points are generated using Latin Hypercube Sampling (LHS), as described earlier in section 2.1.1, which creates a space-filling design within the unit hypercube. The number of initial samples is set to three times the problem's dimensionality ($3D$).

Within this dataset, the trust region is initialized as a hyperrectangle centered around the point with the best observed function value. To focus the search within this trust region, points lying inside the trust region are filtered before performing PCA ensuring the dimensionality reduction is applied only to this local subset.

After performing PCA on the filtered points, which includes weighting using the ranking scheme and centering the points within the trust region the points are projected into a lower-dimensional manifold. The Gaussian Process (GP) model is then trained within this reduced space, and the acquisition function (Log Penalized Expected Improvement) is optimized to select the next candidate point. The bounds of the trust region are also projected into the reduced space to correctly restrict the search.

In terms of the optimization of the acquisition function, the candidate points sampled from the acquisition function in the reduced space are mapped back to the original space. Instead of checking if these points lie within the global domain bounds (as in the original PCA-BO algorithm), we verify if they lie inside the trust region. Points whose counterimage lies outside the trust region are not feasible and are therefore penalized in acquisition value proportional to their distance from the trust region bounds, discouraging sampling outside the region. On the other hand, if the point is feasible, the algorithm calculates the Expected Improvement value of this point. The candidate with the highest (penalized) acquisition value is selected, evaluated, and appended to the list of sampled points. We then update the trust region size and proceed to the next iteration. The main optimization loop runs until the evaluation budget is used up or the trust region becomes too small. In case the trust region shrinks too much, we restart the whole process.

The optimization loop begins by initializing the trust region and continues performing iterations, until either the evaluation budget is exhausted or the trust region size falls below a predefined minimum. If the trust region becomes too small before the budget is exhausted, the algorithm re-initializes the trust to its initial size, new DoE points, and empty history of evaluated points, allowing for multiple local optimization runs within the overall budget.

Each iteration includes filtering points within the trust region, fitting PCA, calculating bounds in the reduced space, transforming points to the reduced space, training GP and optimizing acquisition function, choosing next point to sample based on linear inequality constraints, transform the chosen point from the reduced dimension to the original dimension, evaluate the chosen point and append it to the list of points and their function values. Finally, at the end of each iteration, the trust region is updated and a local DoE point sampling is performed ($1D$) within it unless not enough budget is available, in which case the algorithm terminates.

A few example iterations of LPCA-BO algorithm are visualized in Figure 6, illustrated on a multimodal function F21. In the first iteration, the algorithm explored a misleading region of the search space. However, following a restart, it successfully identified the relevant region, where the PCA-reduced manifold captured the global optimum. Subsequent samples were concentrated near this manifold. Moreover, it can also be seen how after iteration 28, the trust region adjusted its center to be centered around the newly discovered best point.

3.2.1 Trust Region Restart Strategy

The inner optimization loop continues as long as the evaluation budget has not been exhausted and the trust region remains above a minimum size. When the trust region shrinks below a predefined minimum length, the loop exits and triggers the `initialize_restart()` function. Upon restarting, the trust region length is reset to its initial value, and all locally collected points and their corresponding function evaluations are cleared. In addition, success and failure counters are reset, and a new initial design is generated using Latin Hypercube Sampling (LHS) within the full search bounds.

The evolution of the length of the trust region balances two competing objectives. The region must be sufficiently large to contain promising solutions, yet small enough to ensure that the local model remains accurate within its bounds. A restart of the trust region is triggered when it becomes too small to be effective. Specifically, the trust region is considered inefficient when its length falls below a predefined minimum threshold. The restart is triggered when `self.length` $<$ `self.length_min`, where `self.length_min` $= 0.5^7 \approx 0.0078125$.

3.2.2 Trust Region Point Filtering

Before applying PCA, we filter the points to include only those that lie within the current trust region. Let $X \in \mathbb{R}^{n \times D}$ denote the matrix of previously evaluated points, where D is the problem's dimensionality. The trust region T is defined by lower and upper bounds $lb, ub \in \mathbb{R}^D$, such that $T = [ub, lb]$. The algorithm extracts the subset of points $x_i \in X$ that satisfy $x_i \in T$:

$$X_F \leftarrow \{x_i \in X : x_i \in T\}$$

for all i.

If the number of filtered points is less than the minimum required for a stable PCA fitting, which we set to be $max(D, 2)$, we augment this subset X_F by including as many points as needed, based on Manhattan distance to the trust region, to reach the required size.

This filtering procedure is implemented in the `filter_points` method, which returns a boolean mask over X indicating which points are to be included for PCA fitting. This method ensures that PCA has sufficient data to perform meaningful projection, making it robust to temporary sparsity due to stochastic sampling of points and, moreover, the fitted model remains localized to the trust region. The corresponding function values are filtered analogously:

$$y_F \leftarrow \{y_i \in y : x_i \in X_F\}$$

This procedure ensures that sufficient and localized data is always available for fitting the PCA projection reliably, even when sampling sparsity within the trust region is high.

3.2.3 Rank-Based Weighting for PCA Projection

Since the goal of PCA in this context is to identify directions in the original high-dimensional space along which the objective function is most sensitive, it is important to incorporate information about the function values. To do this, a weighting scheme is applied that rescales the data points based on their corresponding function values. By applying the weighting scheme, we are able to adjust the importance of each point. Points with a better function value are associated with a larger weight, moving less compared to points with a worse function value that get a smaller weight assigned. The weighting scheme used is known as the rank-based weighting scheme, where the weight assigned to each data point is calculated as

$$w_i = \ln n \cdot \ln r_i$$

with r_i representing the rank of the i-th point in X based on its function value y. The variable n denotes the total number of data points, and the ranks r_i are assigned in ascending order of the function values. After calculating the pre-weights, we normalize them to ensure they sum to one:

$$\tilde{w}_i = \frac{\tilde{w}_i}{\sum_{i=1}^{n} \tilde{w}_i}$$

This normalization ensures that the total weight is distributed proportionally across all data points. This weighting scheme ensures the weights of less informative points decrease as better points are discovered over iterations, introducing a self-adjusting discount factor. We begin by centering the original data matrix X by subtracting its sample mean. This is achieved by subtracting the mean of all the data points, so the rescaled data becomes: $X' = X - \overline{X}$ where

$$\mu = \frac{1}{n}\sum_{i=1}^{n} X_{i1}, ..., \frac{1}{n}\sum_{i=1}^{n} X_{id} \quad \text{and} \quad \mathbf{1}_n = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}^\top \in \mathbb{R}^n.$$

The calculated weights are stored in a diagonal matrix W, where each diagonal entry corresponds to the weight of a data point. Finally, the weighted and mean-centered data is given by:

$$X' = \overline{X}W.$$

3.2.4 Dimensionality reduction with Principal Component Analysis

After weighting our points, we perform PCA on the weighted points to extract principal components. We obtain a linear map from the high-dimensional space to a lower-dimensional latent space.

3.2.5 Fitting PCA within the Trust Region

We fit Principal Component Analysis (PCA) within the current trust region. The trust region is centered around the current best solution and is bounded in each dimension based on a relative scaling factor. Only points within the trust region are used to compute the PCA projection.

However, when fitting PCA only on the points inside the trust region, the algorithm can easily run into a situation where too few points fall inside the trust region, making the PCA unstable

and meaningless.

Let $X_F \subset X$ denote the set of filtered points lying within the trust region (as defined in Section 3.2.2), and let $y_F \subset y$ be the corresponding function values. If $|X_F| < \max(D, 2)$, the subset is augmented with the closest available points in Manhattan distance until this condition is satisfied.

Once X_F is established, we compute the sample mean of the filtered points:

$$\mu'_F \leftarrow \frac{1}{|X_F|} \sum_{x \in X_F} x.$$

To incorporate information about the objective values into the PCA fitting, we assign weights based on rank. For each $x_i \in X_F$, let $r_i = \text{rank}(y_i)$, where $r_i \in \{1, \ldots, |X_F|\}$ and rank 1 corresponds to the best (lowest) function value. The weight for point i is given by:

$$w_i \leftarrow \frac{\log |X_F| - \log r_i}{\sum_{j=1}^{|X_F|} (\log |X_F| - \log r_j)}.$$

We then center the filtered points with respect to the global mean $\mu \in \mathbb{R}^D$ of all data:

$$\overline{X_F} \leftarrow X_F - \mathbf{1}_{|X_F|}\mu,$$

and rescale the centered matrix using the diagonal weight matrix:

$$W \leftarrow \text{diag}(w_1, \ldots, w_{|X_F|}), \quad X'_F \leftarrow \overline{X_F} W.$$

Finally, PCA is computed on the weighted, centered data:

$$\mu'_F,\ P_r \leftarrow \text{PCA}(X'_F),$$

resulting in projection matrix $P_r \in \mathbb{R}^{D \times r}$ used in subsequent dimensionality reduction steps.

3.2.6 Calculating Reduced-space Bounds

`calculate_reduced_space_bounds` computes bounds in the reduced PCA space that correspond to a trust region defined in the original input space. The function takes as input the trust region bounds `tr_bounds` (a matrix of shape $[D, 2]$, where each row specifies the lower and upper bounds for one of the D dimensions) and a fitted PCA object. It begins by computing the center of the trust region $\mathbf{C}$ as the midpoint between the lower and upper bounds in each dimension. Assuming symmetric bounds, the radius is defined as the distance from the center to either the lower or upper bound. The center point $\mathbf{C}$ is then projected into the reduced PCA space to obtain its reduced representation $\mathbf{C}_r$. Using this center and the radius, the function constructs axis-aligned bounds in the reduced space as $\mathbf{C}_r \pm r$ along each principal component direction. The resulting reduced-space bounds, denoted as `z_bounds`, define the feasible region used for optimization within the lower-dimensional subspace.

3.2.7 Projecting Points to Reduced Space

After fitting PCA on the filtered points, we project these centered points into the reduced subspace $\mathbb{R}^r$ using the top-r principal components P_r that sum to 95% of variance. Given the centered matrix $X_F \in \mathbb{R}^{|F| \times D}$ and their mean $\mu'_F \in \mathbb{R}^D$, the projection is computed as:

$$Z_r \leftarrow P_r(X_F - 1_{|F|}\mu'_F)^\top$$

This operation maps the original points $X_F \in \mathbb{R}^{|F| \times D}$ into the reduced space $Z_r \in \mathbb{R}^{r \times |F|}$. For each point $x_i \in X_F$, the mapping is defined as:

$$z_i = P_r(x_i - \mu'_F)^\top$$

This step enables acquisition optimization in a lower-dimensional space while preserving as much variance (and structure) of the local region as possible.

3.2.8 Training Gaussian Process Regression model and Optimizing Acquisition Function

In each iteration of the algorithm similarly to PCA-BO, a Gaussian Process Regression (GPR) model is trained on the reduced-dimensional dataset to approximate the expensive black-box objective function. Given the reduced inputs Z_r and corresponding function values y, the model produces a predictive mean $\hat{f}$ and variance $\hat{s}^2$:

$$\hat{f}, \hat{s}^2 \leftarrow \mathrm{GPR}(Z_r, y).$$

This surrogate model prediction is then used to guide the selection of the next query point by maximizing an acquisition function. In our implementation, we use the Log Expected Improvement (LogEI) heuristics criterion. To ensure feasibility in the original space, the optimization is constrained such that the reconstructed point lies within the trust region T. A candidate point $z \in S'$ in the reduced space is mapped back to the original space using:

$$x = P_r^\top Z + \mu'_F + \mu,$$

where P_r is the projection matrix, and μ'_F, μ are the means of the filtered and full point sets, respectively.

3.2.9 Acquisition Function: Selecting Next Point Using Penalized Log Expected Improvement

After fitting the Gaussian Process Regression (GPR) model, our algorithm proposes a new candidate point by optimizing an acquisition function in the reduced subspace. Given a candidate point $\mathbf{z} \in \mathbb{R}^r$, it is mapped back to the original high-dimensional space $\mathbb{R}^D$ using the linear map L:

$$L : \mathbb{R}^r \to \mathbb{R}^D, \quad z \mapsto P_r^\top z + \mu' + \mu.$$

where P_r denotes the PCA projection matrix, μ' the local PCA mean, and μ the global data mean used during point pre-processing. Since the linear map L is injective each candidate point z in the reduced space corresponds to a unique point y in the original space. The means μ and μ' are added back since they were previously subtracted from X when mapping to the lower-dimensional space.

After evaluating a candidate point, we augment the dataset in $\mathbb{R}^D$ by appending the new point:

$$X \leftarrow X \cup \{L(z)\}, \quad y \leftarrow y \cup \{y\},$$

and then re-compute the projection matrix (linear map) P_r using the procedure described earlier.

To select the next candidate, the next point to evaluate s selected based on the GP posterior process, balancing $\hat{f}$ with $\hat{s}^2$ (exploitation with exploration). While improving numerical stability in regions where significant improvement is unlikely where standard Expected Improvement (EI) may fail, we use the Single-outcome Log Expected Improvement (LogEI) acquisition function [ADE+25]. This acquisition function is defined as:

$$\mathrm{LogEI}(\mathbf{x}) = \log \mathbb{E}\left[\max(f(\mathbf{x}) - f_{\text{best}}, 0)\right],$$

where f_{best} stands for the best function value observed so far. The expectation is taken with respect to the GP's predictive distribution at point $\mathbf{x}$.

Unlike traditional Expected Improvement (EI), which finds optimization in parallel and multi-objective settings challenging and may suffer from vanishing numerical values in many regions of the search space, LogEI operates in the logarithmic domain, providing better gradient behavior and optimization performance, especially in regions with low probability of improvement. Moreover, as the number of observations, dimensionality, or constraints increases, the challenge becomes even more significant, leading to inconsistent and often sub-optimal performance of standard EI.

LogEI lessens these issues by maintaining numerically stable acquisition value and its gradient even in regions with a low probability of improvement, while retaining optima equivalent to those of its standard EI. Hence, LogEI facilitates stable and effective optimization across various settings.

Points that when projected back end up within the domain are feasible points, however it can also be the case that when we project a point that is inside a domain of the reduced space, but not within the domain of the original space. This is due to the rotational operation performed by PCA. To ensure feasibility, is to apply a penalty $P(x)$ for infeasible points. This penalty is proportional to the degree of infeasibility of the point. If the projected point is outside the problem bounds, we assign a penalty for it when calculating the expected improvement value of a point during sampling of the points for the optimization of the acquisition function.

The acquisition function proposed is penalized expected improvement (PEI). This acquisition function is defined as:

$$\text{LogPEI}(x) = \begin{cases} \mathbb{E}\left[\max\left(0, \min y - f(x)\right) \mid y\right], & \text{if } x \text{ is feasible} \\ \mathbb{E}\left[\max\left(0, \min y - f(x)\right) \mid y\right] - P(x), & \text{if } x \text{ is infeasible} \end{cases} \tag{1}$$

The selected point x^* should be the one, which maximizes the Expected Improvement value. In the calculation of penalty it finds the closest point in the original space to the point in question, calculates its EI value and subtracts the distance from this. However, Penalized Expected Improvement (PEI), does not guarantee that the chosen point is also feasible when projected to the original space. Choosing the closest point in the original space is not the closest point in the reduced space bounds due to the fact that it can be itself outside of the reduced space bounds. Hence, to deal with this problem, if the selected point is outside of the bounds, the point gets clipped to the bounds in order to guarantee that no point outside of the bounds is sampled to avoid encountering a problem of having the center of the trust region outside of the bounds. Although, by doing this x^* might not be the point with the largest LogEI value.

3.2.10 Inverse Mapping and Evaluation

Finally, we map the selected candidate point back to the original space by transforming the selected low-dimensional candidate point back to the original high-dimensional space using the inverse PCA transformation. After this we evaluate the objective function at this new point by adding the new sample and its function value to the dataset. In case termination criterion is not met (e.g., convergence, budget limit), we repeat this loop and start with updating the weighting scheme, reapply PCA, retrain the GPR etc. again. A few representative iterations of our PCA-BO algorithm are visualized in Figure 4, illustrating how the surrogate model, acquisition function, and PCA weighting evolve over the course of the optimization process on a 2D Rastrigin function (F15).

3.2.11 Updating the trust region size

At the beginning of a given local optimization run, we initialize the base side length of the TR, capturing 80 percent of the domain. Every next point candidate is sampled from the reduced space of the trust region. To balance exploration and exploitation, the trust region size is dynamically adjusted throughout the optimization process based on the observed improvement in function evaluations. At the end of each iteration, the newly evaluated function value is compared with the best value found so far. The process is defined inside the `update_trust_region()` function.

The improvement is defined as a decrease in the function value (assuming minimization) by at least a small threshold ϵ:

$$y' < y_{\text{best}} - \epsilon|y_{\text{best}}|$$

with $\epsilon = 10^{-3}$. If the new function value found in the region improves upon the current best by at least ϵ margin, we consider this a successful step. Otherwise, it is counted as a failure. The small threshold ϵ is there to account for potential numerical noise or small fluctuations in function values. If the number of consecutive successful steps reaches a threshold, which we defined as 3, the trust region shrinks by half its size. This corresponds to exploration by widening the region to consider more global area, helping escape a potential local minimum. On the other hand, if the number

of consecutive unsuccessful steps reaches a threshold, which we defined to be 3, the trust region expands by doubling its size, bounded above by a maximum length. This represent exploitation of a promising area, in which the improvement is found.

Following each trust region adjustment, whether shrinking or expanding, we perform additional sampling within the updated trust region boundaries. Specifically, we generate $1 \times D$ new sample points using optimal Latin hypercube sampling, where $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_{n_0}]^\top \in \mathcal{S}^{n_0}$ represents the sample matrix. All sampling operations are conducted subject to computational budget constraints.

3.3 Experiments

We empirically evaluate the performance of both PCA-BO and LPCA-BO by testing both of the agorithms on the multi-modal functions taken from the COCO/BBOB benchmark suite [FHRA09] and compare each of them to the state-of-art PCA-BO. The BBOB problem set includes a wide variety of challenging problems that might resemble real world applications. Specifically, special attention is given to functions with an adequate structure (F15-F19) and functions with weak global structure F20-F24.

Each algorithm is tested on these functions at three different dimensionalities, $D = \{10, 20, 40\}$. For each problem, we set a total evaluation budget of $10D + 50$ function evaluations. The initial Design of Experiments (DoE) consists of $3D$ samples, with the remaining budget allocated to the iterative optimization process. All experiments are repeated over 30 independent runs per function instance, and executed on an ALICE CPU machine. The recorded metrics include the best-so-far function value as a measure of convergence quality and CPU runtime in seconds as a measure of computational efficiency.

Both PCA-BO and LPCA-BO are implemented in Python using the BoTorch farmework. For settings in which we use constraints, at each iteration, the Log Expected Improvement (LogEI) acquisition function is optimized under inequality constraints using the Sequential Least Squares Programming (SLSQP) [MZL22] optimizer, rather than the default Limited-memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS-B) [LN89] algorithm. The optimization is performed within the feasible region defined by the constraints. On the other hand, turning of the constraints we can switch to using Penalized Expected Improvement (PEI) instead. The surrogate model used is a Gaussian Process (GP) with a Matérn 5/2 kernel.

3.3.1 Justifying adjustments for 2D visualizations

In order to produce interpretable visualizations of PCA-BO and LPCA-BO in a 2D setting, the maximum number of principal components is reduced to one. Therefore, the optimization problem substantially increases in its difficulty due to the one principal component capturing significantly less variance than the typical 95% threshold used in high-dimensional settings. In fact, it can get low as 50% variance captured. As a result, the model often fails to fit the data and instead produces nearly flat predictions. To counteract this, we tune the trust region parameters by reducing the initial trust region length to 0.5 and setting the success count threshold required for expansion to 1. These changes encourage more localized and conservative exploration, which helps compensate for

Algorithm 3 Local PCA-assisted Bayesian Optimization (LPCA-BO)

procedure LPCA-BO($f, \mathcal{A}, S$) ▷ f: objective function, $\mathcal{A}$: infill-criterion, S: search space, S': lower-dimensional search space
  while budget not exhausted **do**
    $\ell_{\min} \leftarrow 0.5^7$, $\ell_{\max} \leftarrow 1.6$, $\ell \leftarrow 0.8$ ▷ Reset for trust region restart
    Initialize counters: succcount $\leftarrow 0$, failcount $\leftarrow 0$
    Initialize constants: succtol $\leftarrow 3$, failtol $\leftarrow 3, \epsilon \leftarrow 10^{-3}$
    Create $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_{n_0}]^\top \in S^{n_0}$ with optimal Latin hypercube sampling subject to budget constraints
    $\mathbf{y} \leftarrow (f(\mathbf{x}_1), \ldots, f(\mathbf{x}_{n_0}))^\top$, $n \leftarrow n_0$
    while budget not exhausted **and** $\ell \geq \ell_{\min}$ **do**
      Trust region bounds: $\mathbf{T} \leftarrow [\mathbf{x}_{\text{best}} - \frac{\ell}{2} \cdot (\mathbf{ub} - \mathbf{lb}), \mathbf{x}_{\text{best}} + \frac{\ell}{2} \cdot (\mathbf{ub} - \mathbf{lb})] \cap S$ ▷ $\mathbf{lb}, \mathbf{ub} \leftarrow$ lower and upper bounds of S and $\mathbf{x}_{\text{best}} \leftarrow \mathbf{x}_{i^\star}$ where $i^\star = \arg\min_i y_i$
      Filter points in trust region: $\mathbf{X_F} \leftarrow \{\mathbf{x}_i \in \mathbf{X} : \mathbf{x}_i \in \mathbf{T}\}$
      if $|\mathbf{X_F}| < \max(d, 2)$ **then**
        Augment $\mathbf{X_F}$ with closest points from $\mathbf{X} \setminus \mathbf{X_F}$ until size is $\max(d, 2)$
      end if
      $\mathbf{y_F} \leftarrow \{y_i \in \mathbf{y} : \mathbf{x}_i \in \mathbf{X_F}\}$
      Compute sample mean of all points: $\boldsymbol{\mu} \leftarrow \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i$
      Compute sample mean of filtered points: $\boldsymbol{\mu'_F} \leftarrow \frac{1}{|\mathbf{X_F}|} \sum_{x \in X_F} x$
      Weighting: Let $r_i = \text{rank}(y_i)$ where $r_i \in \{1, \ldots, |X_F|\}$ (1=best), then $w_i \leftarrow \frac{\log |X_F| - \log r_i}{\sum_{j=1}^{|X_F|} (\log |X_F| - \log r_j)}$
      Centering filtered points: $\overline{\mathbf{X_F}} \leftarrow \mathbf{X_F} - \mathbf{1}_{|\mathbf{X_F}|} \boldsymbol{\mu}$
      Rescaling: $\mathbf{W} \leftarrow \text{diag}(w_1, \ldots, w_{|F|})$, $\mathbf{X'_F} \leftarrow \overline{\mathbf{X_F}} \mathbf{W}$
      $\boldsymbol{\mu'_F}, \mathbf{P}_r \leftarrow \text{PCA}(\mathbf{X'_F})$
      Mapping to $\mathbb{R}^r$: $\mathbf{Z}_r \leftarrow \mathbf{P}_r (\overline{\mathbf{X_F}} - \mathbf{1}_{|\mathbf{F}|} \boldsymbol{\mu'_F})^\top$
      GPR training: $\hat{f}, \hat{s}^2 \leftarrow \text{GPR}(\mathbf{Z}_r, \mathbf{y})$
      $\mathbf{z}' \leftarrow \arg\max_{\mathbf{z} \in S'} \text{LogPEI}(\mathbf{z})$
      Mapping to $\mathbb{R}^D$: $\mathbf{x}' \leftarrow \mathbf{P}_r^\top \mathbf{z} + \boldsymbol{\mu'_F} + \boldsymbol{\mu}$
      if $\mathbf{x}' \notin \mathbf{T}$ **then** $\mathbf{x}' \leftarrow \text{clip}(\mathbf{x}', \mathbf{T})$
      $y' \leftarrow f(\mathbf{x}')$
      $\mathbf{X} \leftarrow \mathbf{X} \cup \{\mathbf{x}'\}$, $\mathbf{y} \leftarrow (\mathbf{y}^\top, y')^\top$, $n \leftarrow n + 1$
      if y' improves by $> \epsilon |y_{\text{best}}|$ **then** ▷ TR update (lines 18 - 27)
        succ_count $\leftarrow$ succ_count $+ 1$, fail_count $\leftarrow 0$
      else
        succ_count $\leftarrow 0$, fail_count $\leftarrow$ fail_count $+ 1$
      end if
      if succ_count = succ_tol **then**
        $\ell \leftarrow \min(2\ell, \ell_{\max})$, succ_count $\leftarrow 0$ ▷ Expand trust region
      else if fail_count = fail_tol **then**
        $\ell \leftarrow \ell/2$, fail_count $\leftarrow 0$ ▷ Shrink trust region
      end if
      Additional $1xDimensionality$ sampling within the trust region: $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_{n_0}]^T \in S^{n_0}$ with optimal Latin hypercube sampling subject to budget constraints
    end while
  end while
end procedure

the information loss caused by aggressive dimensionality reduction.

Despite these adjustments, this issue of producing flat GP predictions is still relatively frequent. This issue arises whether the GP is trained using all available data or only the data within the trust region. The problem is even more present when the true objective function is complex and highly non-linear, such as in Gallagher's Gaussian 101-me Peaks Function. In these cases, the reduced subspace lacks sufficient information to enable a reliable GP fit, and the added DoE points often appear as noise since they lie far from the principal component axis.

Moreover, when running more DoE after trust region adjustment in the case of LPCA-BO, many of the new DoE points make the GPR model fitting very hard because they are much more likely to appear as noise because they are not on the pc line and instead somewhere completely different in space.

A potential solution could be increasing the noise variance of the GP to improve robustness to the outlier-like behavior of off-manifold points. However, this issue may be inherent to low-dimensional visualizations as projecting complex high-dimensional landscapes into 1D leads to unavoidable information loss. Therefore, these challenges should be interpreted as limitations of 2D demos rather than fundamental problems of the algorithm.

Another important consideration in the 2D visualizations is the mismatch between the principal component direction, in this case PC1, and the actual feasible domain defined by the bounds. The acquisition function optimization is performed along the full extent of the PC1 line, under the assumption that this represents the reduced search space. However, the actual legal area that lies within the bounds can sometimes not correspond to the length of the PC1 line. As a result, this is not always accurate as the true feasible region in the reduced space is actually the portion of the PC1 line that intersects with the box constraints (Figure 7). In summary, the true reduced space bounds area, which satisfies the constraints represented by the intersection between PC and bounds, is not always corresponding to the length of PC1, which is considered for the acquisition function optimization.

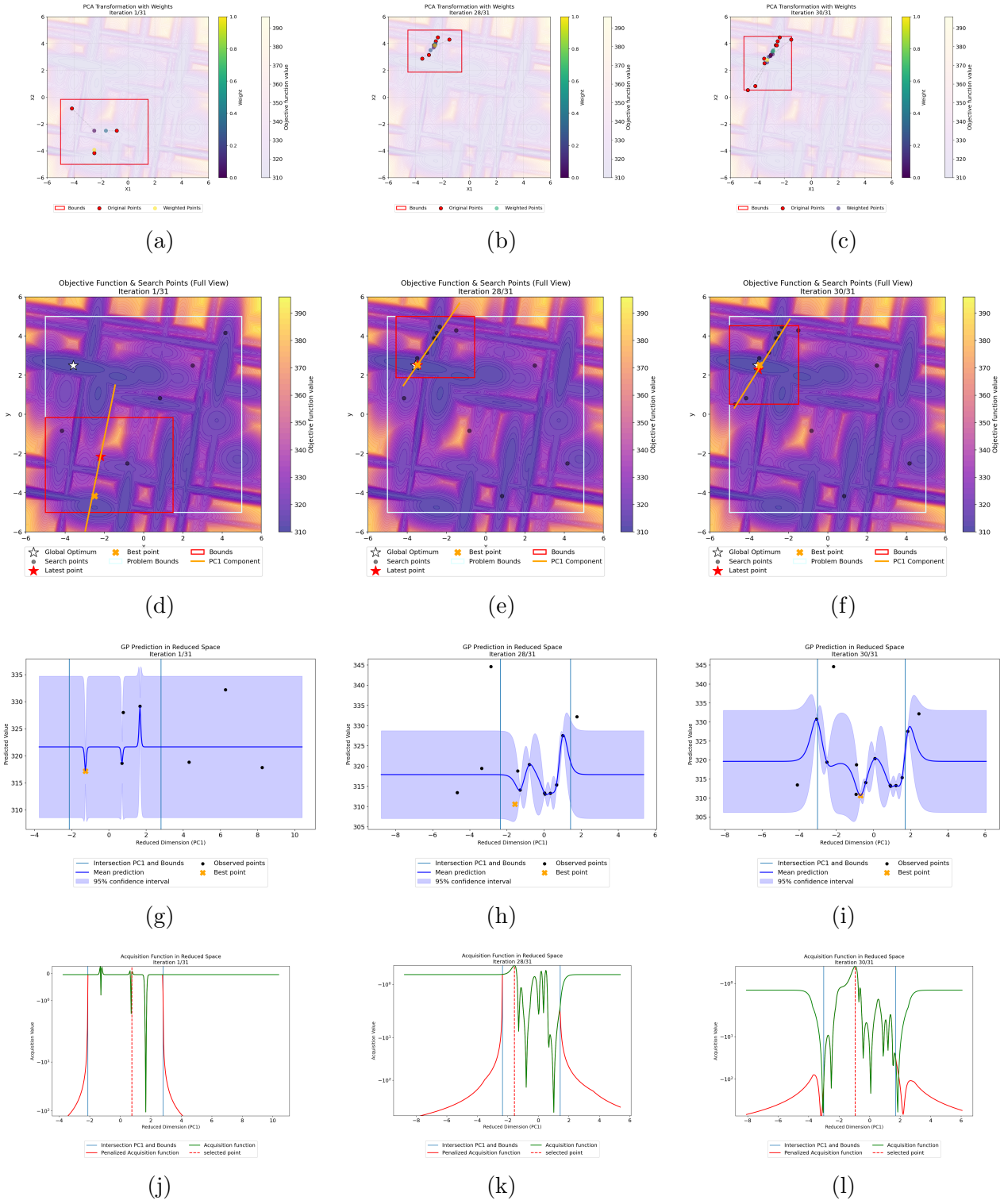


Figure 6: Visualization of the 1st, 28th, and 30th iteration of the LPCA-BO algorithm applied to a 2D Gallagher 101 Peaks function (BBOB F21). Each column corresponds to one iteration. Rows represent: (a-c) the weighting of points used for fitting the PCA line, (d-f) the true objective function landscape with sampled points, (g-i) the fitted Gaussian Process (GP) models, and (j-l) the corresponding acquisition function (Penalized LogEI) landscape. A complete example of this run, along with additional examples illustrating the behavior of the algorithm on various BBOB functions, is available in the accompanying Github repository.

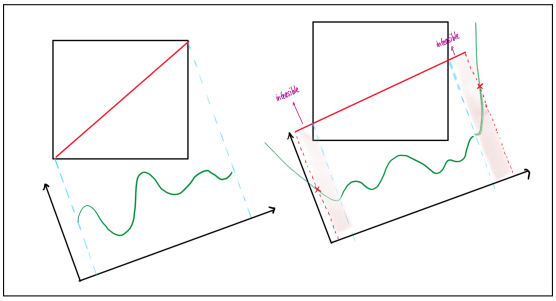


Figure 7: A simple sketch of the mismatch between the intersection of the PCA line with the bounds and the feasible sampling region.

4 Results

We now present the performance outcomes of PCA-BO and LPCA-BO, evaluated on the functions from the COCO/BBOB benchmark suite, with a focus on two categories of multi-modal functions with adequate global structure (F15–F19) and those with weak global structure (F20–F24). For each setting, we report the average best-so-far function value as a measure of convergence quality (Figure 8, Figure 9, Figure 10) and the average CPU runtime as an indicator of computational efficiency (Figure 11). All results are averaged over 30 independent runs. Each algorithm is tested in dimensionality 10, 20 and 40. Performance comparisons are made against PCA-BO implementation as a baseline, allowing us to quantify the improvements achieved by the proposed LPCA-BO approach.

In 10 dimensions, LPCA-BO demonstrates improved performance over PCA-BO across all benchmark functions, with the exception of the two multimodal cases F19 and F24. On the high-conditioning unimodal function F18, both methods exhibit nearly identical convergence behavior. A similar trend is observed in 20 dimensions: LPCA-BO consistently outperforms PCA-BO except on F19 and F24, while the convergence on F18 remains almost identical. Function F11, which shows high variance under PCA-BO in 20 dimensions, shows greater stability in the 40-dimensional case, but with almost identical convergence rate. In the 40-dimensional setting, LPCA-BO continues to outperform PCA-BO on all functions except F19 and F24. Across all three problem spaces, the performance gap between the two algorithms tends to widen as the dimensionality of the search space increases. This suggests that LPCA-BO scales more effectively and exhibits greater robustness in high-dimensional optimization problems.

In terms of CPU time, the trust region approach is faster compared to PCA-BO and scales better in higher dimensions. Figure 11 shows the CPU time (in seconds) required by PCA-BO (blue) and LPCA-BO (red) across all 24 test functions for three different dimensions: 10D, 20D, and 40D. Overall, across all three dimensions, LPCA-BO consistently shows lower median CPU times compared to PCA-BO. This computational advantage becomes more pronounced as dimensionality increases. The time difference is particularly evident in 20D and especially in 40D, where PCA-BO often incurs substantially higher runtimes. The spread of the box plots also indicates reduced variance in LPCA-BO's runtimes at higher dimensions, suggesting more stable and predictable performance as the problem dimensionality increases.

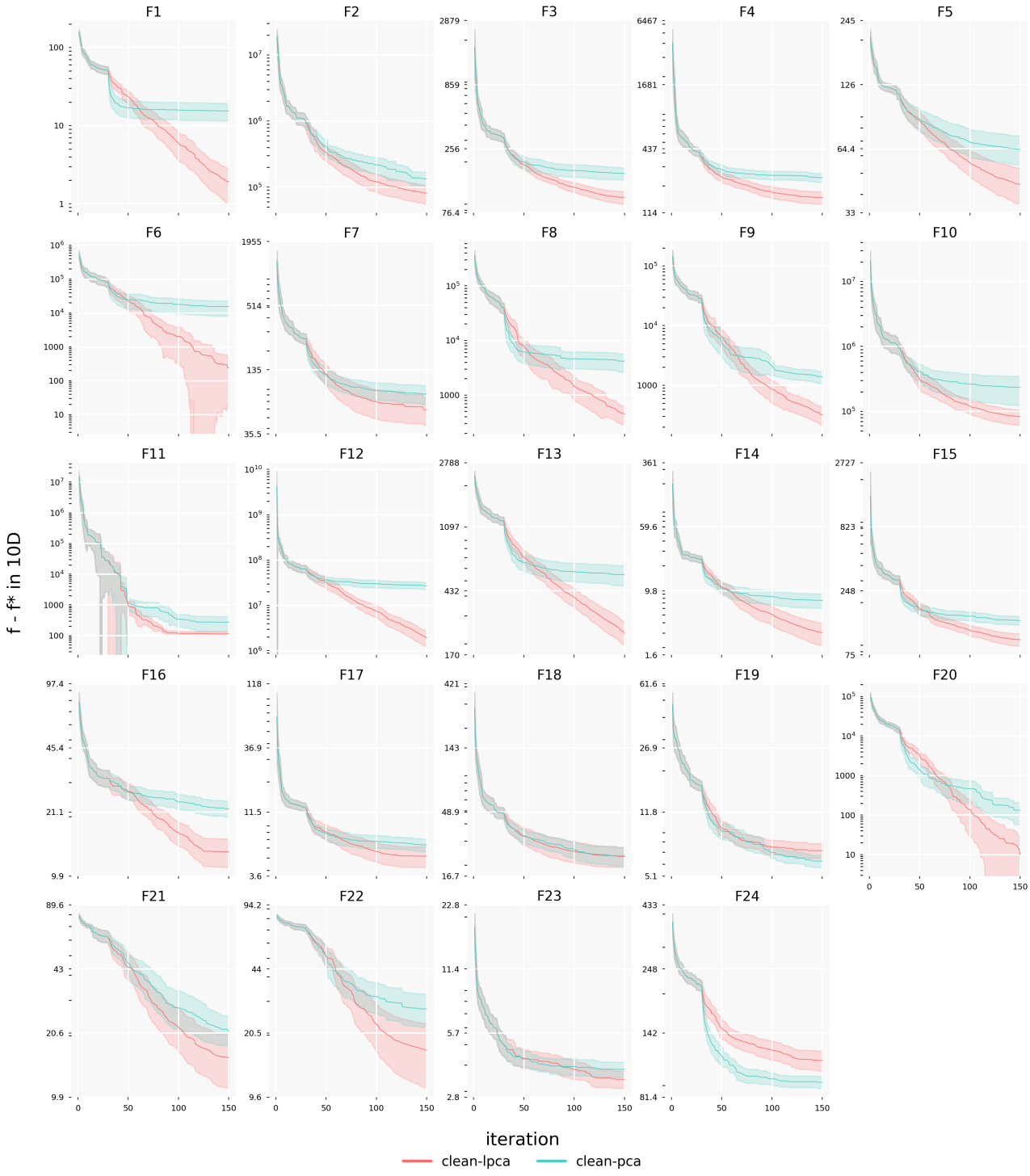


Figure 8: Average target precision (relative to the optimum) as a function of iteration count for BO (red) and PCA-BO (blue), evaluated on the functions from the BBOB benchmark suite in 10D. The results are based on 30 independent runs, with shaded regions denoting 95% confidence intervals. Iteration count corresponds to the number of function evaluations including the initial design (DoE).

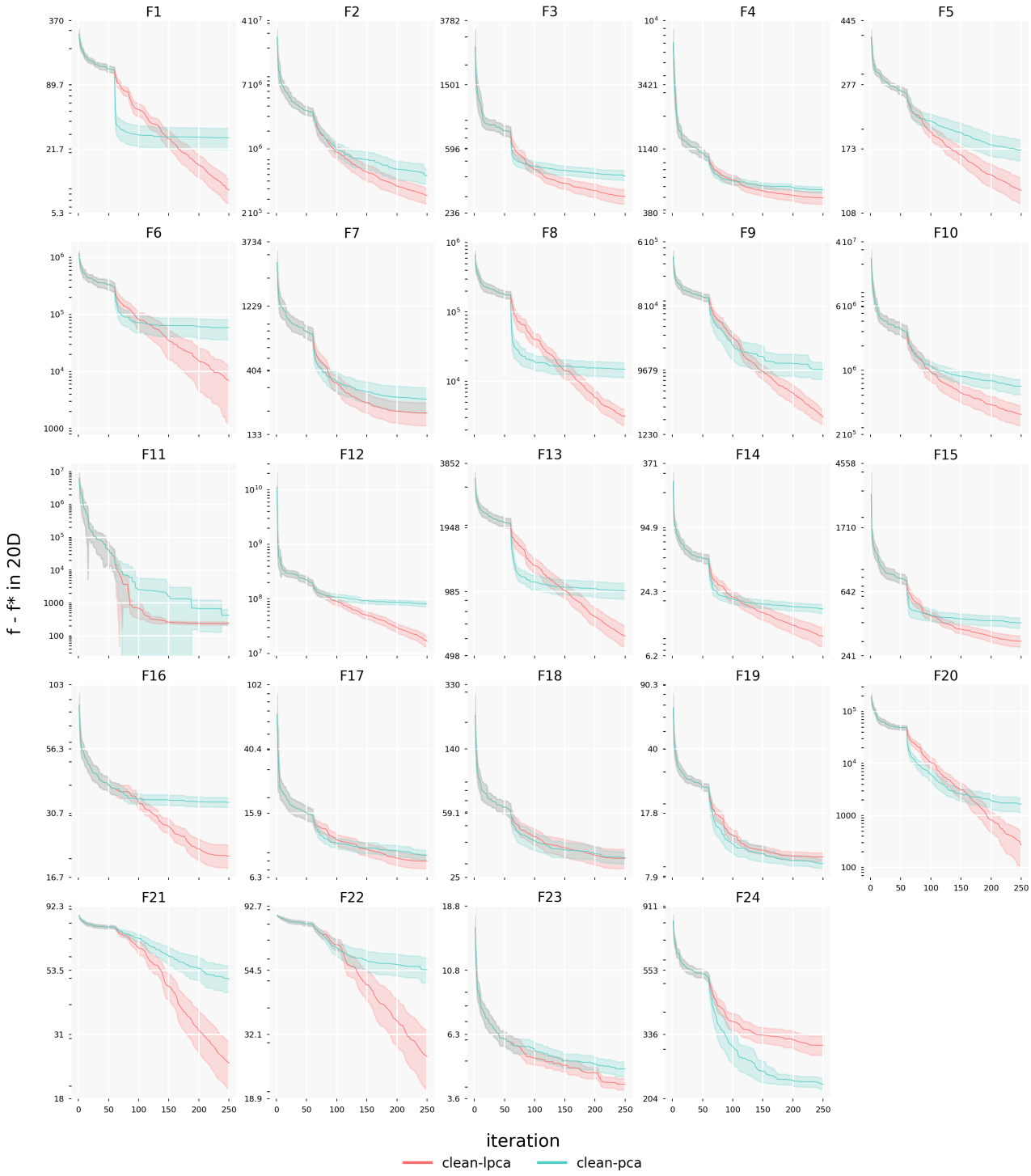


Figure 9: Average target precision (relative to the optimum) as a function of iteration count for BO (red) and PCA-BO (blue), evaluated on the functions from the BBOB benchmark suite in 20D. The results are based on 30 independent runs, with shaded regions denoting 95% confidence intervals. Iteration count corresponds to the number of function evaluations including the initial design (DoE).

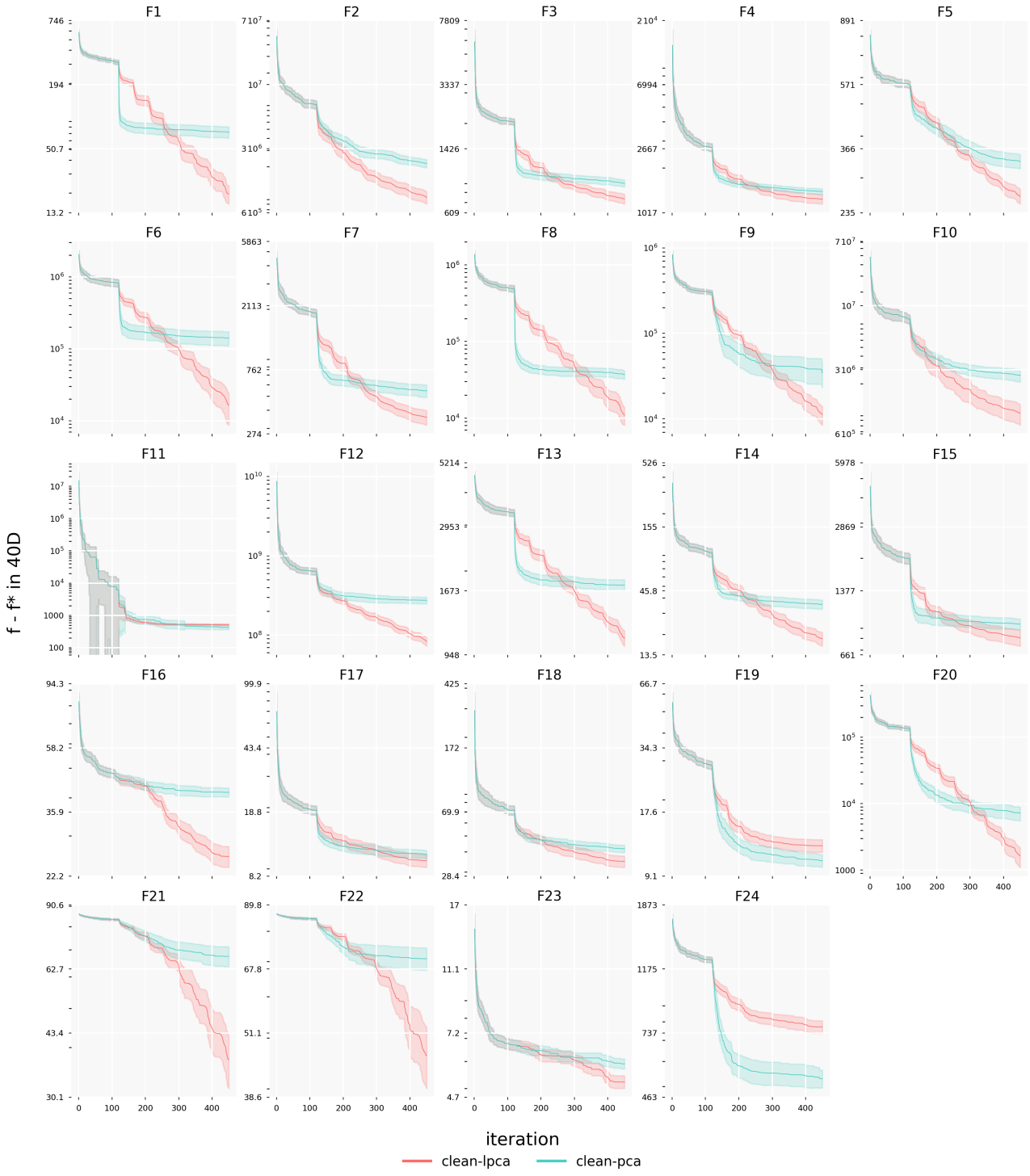


Figure 10: Average target precision (relative to the optimum) as a function of iteration count for BO (red) and PCA-BO (blue), evaluated on the functions from the BBOB benchmark suite in 40D. The results are based on 30 independent runs, with shaded regions denoting 95% confidence intervals. Iteration count corresponds to the number of function evaluations including the initial design (DoE).

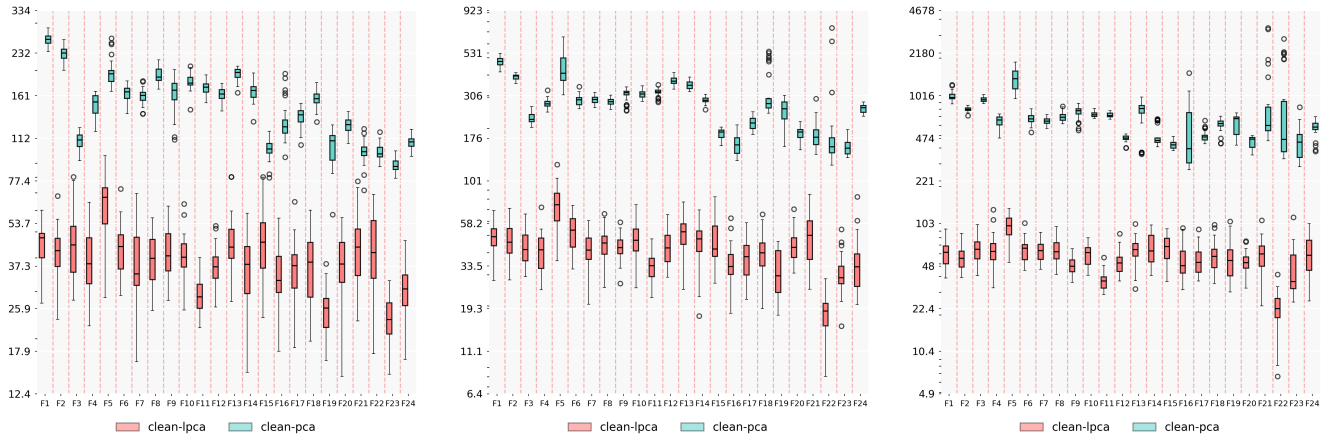


Figure 11: Comparison of CPU time (in seconds) for PCA-BO (blue) and LPCA-BO (red) across three different problem dimensions (from left to right: 10D, 20D, and 40D).

5 Discussion and Future Research

The overall improvement on all functions can be explained by performing PCA only on samples within the trust region, capturing the local manifold structure more accurately and identifying search directions that are relevant to that particular subregion. The search is more focused, eliminating potentially irrelevant samples that could mislead the PCA projection. Trust regions naturally provide a mechanism for balancing local exploitation (region shrinkage) with exploration (region expansion). This dynamic adaptation prevents the algorithm from getting stuck with outdated global projections. The advantage is particularly visible in multimodal functions, where global PCA might average across multiple basins of attraction, preventing capturing meaningful directions. In such settings, the algorithm is more likely to escape from local optima as the trust region size iteratively adapts and helps with PCA projection reveal new promising directions that global PCA might miss, avoiding possible convergence plateaus. Additionally, the incorporation of randomly sampled points after every trust region update contributes to exploration by enabling the trust region to recenter around newly identified promising area, even if it lies outside the current PCA subspace.

The consistent underperformance of LPCA-BO on F19 and F24 reveals fundamental limitations of localized search on specific landscape types. Function F19 (Griewank-Rosenbrock F8F2) is a highly nonlinear, non-separable, and non-convex function characterized by a curved and narrow valley, flat regions, and numerous local optima. This landscape challenges LPCA-BO, as its localized search mechanism can limit exploration. In particular, the success threshold for trust-region updates (set to 3) may need to be lowered to allow more frequent exploration. Once the algorithm exits the valley, it may struggle to re-enter it and effectively explore promising regions.

On the other hand, multimodal function F24 (Lunacek bi-Rastrigin) is characterized by a double-funnel structure. The global landscape consists of two major basins of attraction (funnels), with a superimposed cosine modulation introducing many local optima. The global optimum lies in

a narrow basin, while a broader local optimum occupies approximately 70% of the search space volume within the domain $[-5, 5]^D$. The cosine modulation introduce many local optima. To realize the search is happening in a wrong funnel, the algorithm needs to escape this local minimum by expanding more. As a result, localized methods such as trust-region-based approaches can struggle, as they can become trapped in the dominant local basin and fail to explore globally. Again, this might be due to too high success count threshold.

Additionally, function F11 (Discus function) is a unimodal function with high conditioning, meaning that optimization requires careful navigation along steep and narrow valleys. In 20 dimensions, the projection introduced by PCA-BO may intermittently capture misleading directions due to a limited number of informative samples, resulting in inconsistent performance and high variance. As dimensionality increases to 40, the curse of dimensionality may cause the PCA projection to become increasingly dominated by noise or misleading variance in the data, leading the optimizer to settle into a more consistent but suboptimal search trajectory. In higher dimensions, PCA-BO may fail to align the principal components with these sensitive directions due to the global nature of its projection, especially when most directions contribute equally little to variance. As a result, while the optimization appears more stable in terms of variance, it may fail to exploit the true landscape of the objective function, thus preventing further convergence.

Overall, the observed improved scalability of LPCA-BO compared to PCA-BO, seen as emphasized improvements in higher dimensional space compared to lower dimensional space, might be due to the algorithm's use of localized projections and trust regions better capturing relevant local structures, making search in complex, high-dimensional landscapes more efficient.

Regarding CPU times, incorporating additional design-of-experiment (DoE) points between trust region updates may further improve the efficiency of LPCA-BO. However, increasing the number of DoE samples could also influence the dimensionality reduction process. Since DoE points are sampled across the full original space, the resulting PCA may require a larger number of principal components to meet the variance retention threshold. This, in turn, could reduce the effectiveness of the dimensionality reduction and increase computational cost per BO iteration, potentially offsetting the intended speed-up. However, by adding more DoE, we perform less BO iterations, therefore, the algorithm takes less time.

Future research could explore whether the localization principles applied here can be extended to nonlinear dimensionality reduction techniques, such as Kernel PCA (KPCA), which may better capture nonlinear correlations in complex landscapes such as the curved valleys that challenge LPCA-BO in F19. It would be interesting to combine this approach with other methods such as sampling points in the orthogonal space of the reduced dimension space, which could help with exploration beyond the reduced space. Another potential direction involves refining the PCA fitting procedure, for instance by employing a truncation scheme that selects a fixed percentage of the best-performing samples rather than using a weighting scheme [RWB+20]. Furthermore, parameter tuning, such as adjusting the initial and minimum trust region sizes, success and failure thresholds, could help optimize the algorithm's performance. Moreover, adding multiple trust regions would also be an interesting way to explore the possibilities of LPCA-BO and its performance on challenging multimodal functions by allowing concurrent local searches. Possibly replacing PCA-BO's

acquisition function with TuRBO's Thompson sampling would also be an interesting direction to explore and compare. It would also be valuable to conduct more extensive testing, particularly in high-dimensional settings, where challenges differ significantly from those in low-dimensional spaces, for example in terms of GP training or acquisition function optimization.

6 Conclusion

Our results show improvements in both CPU time and convergence rates. In conclusion, incorporating trust regions and localized dimensionality reduction improves the efficiency and reliability of PCA-BO, particularly in settings with non-uniform variance across parameters or anisotropic basins of attraction. By applying PCA within dynamically updated trust regions, LPCA-BO captures the local structure of the search space more effectively, aligning search directions with locally informative dimensions while avoiding misleading global variance. The dynamic nature of the trust region allows the algorithm to adapt over time, balancing exploitation within known regions and exploration into new areas as the region expands and re-centers. This prevents stagnation associated with outdated global projections, a critical factor in escaping local optima in multimodal functions.

However, while LPCA-BO shows consistent improvements and better scalability in higher dimensions, limitations remain on complex multimodal functions (F19 and F24). Further tuning of trust region parameters, various other adjustments in the algorithm and additional testing to evaluate the impact of such adjustments may be necessary to overcome these challenges.

References

[ADE+25] Sebastian Ament, Samuel Daulton, David Eriksson, Maximilian Balandat, and Eytan Bakshy. Unexpected improvements to expected improvement for bayesian optimization, 2025.

[ARD] *Automatic Relevance Determination*, chapter Chapter 4, pages 59–75.

[EPG+20] David Eriksson, Michael Pearce, Jacob R Gardner, Ryan Turner, and Matthias Poloczek. Scalable global optimization via local bayesian optimization, 2020.

[FHRA09] Steffen Finck, Nikolaus Hansen, Raymond Ros, and Anne Auger. Real-parameter black-box optimization benchmarking 2009: Noiseless functions definitions. Technical Report RR-6829, INRIA, 2009. Updated version as of February 2019.

[FLS05] Kai Tai Fang, Runze Li, and Agus Sudjianto. *Design and modeling for computer experiments*. CRC Press, October 2005. Publisher Copyright: © 2006 by Taylor Francis Group, LLC. All rights reserved.

[F.R01] Karl Pearson F.R.S. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

[Fra18] Peter I. Frazier. A tutorial on bayesian optimization, 2018.

[GKD19] Jochen Görtler, Rebecca Kehlbeck, and Oliver Deussen. A visual exploration of gaussian processes. *Distill*, 2019. https://distill.pub/2019/visual-exploration-gaussian-processes.

[LN89] D. C. Liu and J. Nocedal. *On the Limited Memory BFGS Method for Large Scale Optimization*, pages 503–528. Mathematical Programming, 1989.

[MZL22] Yingjie Ma, Nan Zhang, and Jie Li. Improved sequential least squares programming–driven feasible path algorithm for process optimisation. In Ludovic Montastruc and Stephane Negny, editors, *32nd European Symposium on Computer Aided Process Engineering*, volume 51 of *Computer Aided Chemical Engineering*, pages 1279–1284. Elsevier, 2022.

[Rai22] Tom Rainforth. Chapter 7: Bayesian optimization, 2022.

[RWB+20] Elena Raponi, Hao Wang, Mariusz Bujny, Simonetta Boria, and Carola Doerr. High dimensional bayesian optimization assisted by principal component analysis, 2020.

[Sob67] I.M Sobol'. On the distribution of points in a cube and the approximate evaluation of integrals. *USSR Computational Mathematics and Mathematical Physics*, 7(4):86–112, 1967.

[SSW+16] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando de Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.