# LOCAL PREFERENTIAL BAYESIAN OPTIMIZATION

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Bayesian optimization (BO) is a popular and effective approach for tuning expensive, noisy experiments, but requires the formulation of an explicit objective function. Preferential BO (PBO) removes this requirement by learning from pairwise human feedback, yet existing methods struggle to efficiently optimize beyond low- and medium-dimensional problems due to their global search approaches. We address this limitation by developing a family of local PBO methods that transfer key ideas from high-dimensional BO to the preferential setting. In particular, we introduce local PBO methods that adapt trust-region and derivative-informed local search to pairwise preference feedback, where the latter exploits first- and second-order derivatives of the Laplace-approximated Gaussian process (GP) posterior. To reduce sensitivity to initialization, we additionally consider global-to-local variants that transition from global exploration to local refinement. Our benchmarks on GP sample paths, standard optimization benchmark functions, and policy-search tasks show that local PBO methods are especially effective in high-dimensional and complex landscapes with steep optima. Compared with global preference-based baselines, they can substantially reduce cumulative regret, making them particularly useful for real-world preference-based optimization tasks such as policy search.

## 1 INTRODUCTION

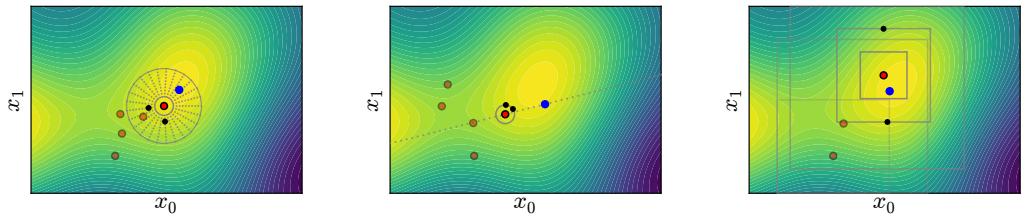
Bayesian optimization (BO) (Garnett, 2023) is widely used to tune expensive black-box systems, including controller parameters (Neumann-Brosig et al., 2019; Coutinho et al., 2024; Holzapfel et al., 2024; Menn et al., 2024), industrial processes (De Blasi et al., 2024; Zimmermann et al., 2025), and policies for robotic systems (He et al., 2025; Hose et al., 2025). However, many such applications do not admit a natural scalar objective: performance may depend on qualitative user feedback based on criteria such as smoothness or comfort. Preferential Bayesian optimization (PBO) addresses this issue by replacing numeric objective evaluations with pairwise comparisons (González et al., 2017), making it particularly well suited to real-world optimization tasks and human feedback.

Despite its practical appeal, existing PBO methods have primarily been developed and applied in low- and medium-dimensional settings (Astudillo et al., 2023; Xu et al., 2024; González et al., 2017). In high-dimensional problems, the global search strategies of these state-of-the-art methods can become inefficient due to the rapidly growing search space (Müller et al., 2021; Wu et al., 2023). Standard BO often addresses such settings through local search approaches, such as trust-region BO methods (Eriksson et al., 2019) and gradient-based BO methods (Müller et al., 2021; Nguyen et al., 2022; Tang et al., 2026; He et al., 2025; Brunzema & Trimpe, 2025). This is especially attractive when cumulative regret matters in addition to final performance: in real-world applications such as controller tuning, poor intermediate evaluations can be costly (Menn et al., 2026).

In this paper, we revisit PBO through the lens of local search. Rather than treating PBO as a purely global search problem, we develop a local view in which pairwise comparisons are used to progressively refine promising regions of the design space. Building on this perspective, we adapt two central ideas from high-dimensional BO, namely trust-region optimization (Eriksson et al., 2019) and derivative-informed (Müller et al., 2021; Brunzema & Trimpe, 2025) local refinement, to the preferential setting (Figure 1). The resulting algorithms form a new family of local PBO methods. Among them, we introduce, to the best of our knowledge, the first PBO methods that leverage higher-order information from the Laplace-approximated Gaussian process (GP) posterior, including both first- and second-order variants. Since local gradient-based methods can be sensitive



Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.




(a) GIPBO (b) PrefSQP (c) TuRPBO

Figure 1: Overview of the local methods with 15 evaluations per plot. In GIPBO and PrefSQP, ● marks the gradient-estimation center; in TuRPBO, it marks the trust-region center. ● marks previous evaluations, ● the next evaluation, ○ search bounds, and □ the trust region. (a) GIPBO selects points within the search bounds to minimize gradient uncertainty at the center, then takes a fixed-step gradient update. (b) PrefSQP estimates the gradient and Hessian by random search within the bounds, followed by a line search in the descent direction. (c) TuRPBO adapts the trust-region size heuristically.

to initialization, we also evaluate global-to-local (G2L) variants that first identify a promising region through global search before switching to local refinement.

Our experiments on GP sample paths, standard optimization benchmark functions, and policy-search tasks show that the proposed local PBO methods are particularly effective in high-dimensional and complex landscapes with steep optima. While global methods remain competitive in smoother and lower-dimensional settings, local approaches often achieve lower cumulative regret by efficiently refining promising regions. These results make local PBO methods especially useful for real-world preference-based optimization tasks.

**Contributions** In summary, the key contributions of this paper are:

- **C1** We introduce a new family of local PBO methods that bring trust-region optimization and derivative-informed local refinement to pairwise preference-based optimization.
- **C2** With GIPBO and PrefSQP, we develop the first derivative-informed PBO methods, which exploit first- and second-order posterior predictive information from the Laplace-approximated pairwise GP posterior.
- **C3** We benchmark local PBO, including global-to-local warm-started variants, against global preference-based baselines on GP sample paths, standard benchmark functions, and reinforcement-learning policy-search tasks, showing that local methods are especially effective in complex high-dimensional landscapes and can reduce cumulative regret when good initial regions are available.

## 2 Problem Formulation

We consider the maximization of an unknown function $f : \Theta \to \mathbb{R}$ over a compact set $\Theta \subseteq \mathbb{R}^d$:

$$\boldsymbol{x}^* \in \arg\max_{\boldsymbol{x}\in\Theta} f(\boldsymbol{x}). \tag{1}$$

Instead, at each iteration $n \in \mathbb{I}_T \coloneqq \{1, \dots, T\}$, where $T$ is the total query budget, an algorithm queries a pair of inputs $(\boldsymbol{x}_n, \boldsymbol{x}'_n) \in \Theta^2$ and observes only a binary preference. Specifically, the underlying noisy evaluations of the unknown utility function are given by

$$y_n = f(\boldsymbol{x}_n) + \epsilon_n, \qquad y'_n = f(\boldsymbol{x}'_n) + \epsilon'_n, \qquad \epsilon_n, \epsilon'_n \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0, \sigma_{\text{n}}^2). \tag{2}$$

The observation is a binary random variable $\pi_n \in \{0, 1\}$ indicating which noisy evaluation is larger:

$$\pi_n = \begin{cases} 0, & \text{if } y_n > y'_n, \\ 1, & \text{otherwise.} \end{cases} \tag{3}$$

Equivalently, each observation can be represented as an ordered comparison $(\boldsymbol{x}_n^+, \boldsymbol{x}_n^-)$, where $\boldsymbol{x}_n^+$ denotes the preferred input and $\boldsymbol{x}_n^-$ the non-preferred input.

**Goal of the Paper** We aim to solve (1) in high-dimensional settings using only noisy pairwise comparisons (3). To this end, we propose local PBO methods that augment PBO with state-of-the-art local BO mechanisms. Specifically, we characterize the changes needed to enable the use of these mechanisms with pairwise observations and to evaluate when local PBO improves over existing global preference-based optimization methods.

## 3 Related Work

**Preferential Bayesian Optimization** Existing PBO methods are primarily built on the pairwise GP model of (Chu & Ghahramani, 2005), combined with global acquisition strategies. Early work pairs this surrogate with Thompson sampling (TS) (González et al., 2017), as implemented in the POLAR framework (Tucker et al., 2022). More recent methods use the decision-theoretic acquisition function EUBO (Lin et al., 2022) and its batch extension qEUBO (Astudillo et al., 2023), which outperform TS approaches and are available in BoTorch (Balandat et al., 2020). POP-BO (Xu et al., 2024) provides regret guarantees, but relies on direct function evaluations for hyperparameter selection and is therefore not applicable in a strictly preference-based setting (Erarslan et al., 2025). Beyond pairwise GPs, the skewed GP approximation (Benavoli et al., 2021) underlies the hallucination believer method (Takeno et al., 2023), which can be used with standard acquisition functions such as UCB and EI, while parametric, non-Bayesian methods like GLISp (Bemporad & Piga, 2021) model the utility as a linear combination of radial basis functions. Despite these advances, all state-of-the-art PBO methods rely on global search and therefore inherit the same inefficiencies as standard BO in high dimensions.

**Scaling PBO to High Dimensions** High-dimensional PBO has mainly been addressed through subspace reduction: LineCoSpar (Tucker et al., 2020) uses the mechanism of Kirschner et al. (2019) to scale up to six dimensions, Zhang et al. (2023) exploit the assumption that effective dimensionality is lower than the ambient one, and Mikkola et al. (2020) similarly employ projective subspaces. These methods all assume the preference structure lies in a low-dimensional subspace, which may not hold in complex settings. An alternative direction is slider-based PBO (Koide & Okoso, 2024), which balances local exploitation and global exploration via EUBO, but its feedback mechanism may not generalize and no implementation is publicly available. Overall, PBO methods that operate directly in the original high-dimensional space remain limited; this paper proposes new ideas for scaling PBO.

**Local Bayesian Optimization** While subspace projections have also been explored in standard BO (Kirschner et al., 2019), restricting the search to local subregions has emerged as a key mechanism for high-dimensional problems, achieving substantially improved sample efficiency by prioritizing local refinement over global exploration. Prominent approaches follow two paradigms: trust-region methods (Eriksson et al., 2019; Eriksson & Poloczek, 2021; Yi et al., 2024) that iteratively restrict the search to dynamically adapted local regions, and gradient-informed methods such as GIBO (Müller et al., 2021) and subsequent variants (Nguyen et al., 2022; Tang & Paulson, 2024; He et al., 2025; Fan et al., 2024; von Rohr et al., 2024; Ip et al., 2025) that exploit local gradient information of the surrogate. More recently, BayeSQP (Brunzema & Trimpe, 2025) extends the gradient-based line by additionally incorporating second-order information. Motivated by the promise of these local approaches and the lack of scalable PBO methods, we adapt trust-region and derivative-informed local BO mechanisms to the preferential setting, giving rise to a new family of local PBO methods.

## 4 Preference Modeling with Pairwise Gaussian Processes

Our local PBO methods use pairwise GPs as surrogate models, which are standard in PBO (González et al., 2017; Tucker et al., 2022; Lin et al., 2022; Astudillo et al., 2023; Xu et al., 2024). We first review the pairwise GP model following (Chu & Ghahramani, 2005; Brochu et al., 2010), and then derive the approximate joint conditional distribution of the latent function value, its gradient, and its Hessian under the Laplace-approximated posterior. These quantities provide the predictive information required by our derivative-informed local PBO methods GIPBO and PrefSQP. Since this is the first work to use first- and second-order posterior predictive information in PBO, we explicitly formalize these quantities and their assumptions.

**Prior** Following Chu & Ghahramani (2005); Brochu et al. (2010), we model the latent utility with a zero-mean GP. Let $\boldsymbol{X} = [\boldsymbol{x}_i]_{i=1,\dots,P}$ denote the unique input locations appearing in the pairwise comparisons and $\boldsymbol{f} = [f(\boldsymbol{x}_1), \dots, f(\boldsymbol{x}_P)]^\top$. Then

$$\boldsymbol{f} \sim \mathcal{N}(0, \boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}}), \qquad \boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}} \coloneqq K(\boldsymbol{X}, \boldsymbol{X}),$$

where the $ij$-th entry of $\boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}}$ is $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$.

**Likelihood** For a comparison between $\boldsymbol{x}_n$ and $\boldsymbol{x}'_n$, we observe only which noisy output is larger, where $y_n = f(\boldsymbol{x}_n) + \epsilon_n$, $y'_n = f(\boldsymbol{x}'_n) + \epsilon'_n$, and $\epsilon_n, \epsilon'_n \sim \mathcal{N}(0, \sigma_{\mathrm{n}}^2)$. Writing the preferred and non-preferred inputs as $(\boldsymbol{x}_n^+, \boldsymbol{x}_n^-)$ gives the probit likelihood

$$p(\boldsymbol{x}_n^+ \succ \boldsymbol{x}_n^- \mid \boldsymbol{f}) = \Phi(z_n), \qquad z_n = \frac{f(\boldsymbol{x}_n^+) - f(\boldsymbol{x}_n^-)}{\sqrt{2}\sigma_{\mathrm{n}}}. \tag{4}$$

For $\mathcal{D} = \{(\boldsymbol{x}_n^+, \boldsymbol{x}_n^-)\}_{n=1}^N$, the likelihood factorizes as

$$p(\mathcal{D} \mid \boldsymbol{f}) = \textstyle\prod_{n=1}^N \Phi(z_n). \tag{5}$$

**Posterior** Since (5) is non-Gaussian, we follow Chu & Ghahramani (2005) and use a Laplace approximation around the MAP estimate

$$\hat{\boldsymbol{f}} = \arg\min_{\boldsymbol{f}} \left[ -\sum_{n=1}^N \log \Phi(z_n) + \frac{1}{2} \boldsymbol{f}^\top \boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}}^{-1} \boldsymbol{f} \right]. \tag{6}$$

The Hessian of the negative log posterior at $\hat{\boldsymbol{f}}$ is $\boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}}^{-1} + \boldsymbol{C}$, where, letting $\boldsymbol{h}_n$ contain $+1$ at $\boldsymbol{x}_n^+$, $-1$ at $\boldsymbol{x}_n^-$, and 0 otherwise,

$$\boldsymbol{C} = \sum_{n=1}^N \frac{1}{2\sigma_{\mathrm{n}}^2} \left[ \left( \frac{\varphi(z_n)}{\Phi(z_n)} \right)^2 + z_n \frac{\varphi(z_n)}{\Phi(z_n)} \right]_{\hat{\boldsymbol{f}}} \boldsymbol{h}_n \boldsymbol{h}_n^\top. \tag{7}$$

Thus,

$$p(\boldsymbol{f} \mid \mathcal{D}) \approx q(\boldsymbol{f} \mid \mathcal{D}) = \mathcal{N}\left( \hat{\boldsymbol{f}}, \left( \boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}}^{-1} + \boldsymbol{C} \right)^{-1} \right). \tag{8}$$

Unlike standard GP regression with Gaussian observation noise, $\boldsymbol{C}$ depends on the comparison outcomes through $\hat{\boldsymbol{f}}$. This dependence is important when actively minimizing gradient uncertainty in Sec. 5.1.

**Joint Conditional Distribution** To enable derivative-informed optimization, analogous to local BO methods in the scalar-feedback setting (Müller et al., 2021; Brunzema & Trimpe, 2025), we require posterior information about $f$, its gradient, and its Hessian. For a sufficiently differentiable GP prior, function values and derivatives are jointly Gaussian (Rasmussen & Williams, 2006). Combined with the Gaussian Laplace approximation above, this yields the approximate joint predictive distribution

$$\left( \begin{bmatrix} f(\boldsymbol{x}) \\ \nabla f(\boldsymbol{x}) \\ \mathrm{vec}\{\nabla^2 f(\boldsymbol{x})\} \end{bmatrix} \,\middle|\, \boldsymbol{X}, \mathcal{D} \right) \approx \mathcal{N}\left( \begin{bmatrix} \mu_f^{\mathrm{pref}}(\boldsymbol{x}) \\ \boldsymbol{\mu}_{\nabla f}^{\mathrm{pref}}(\boldsymbol{x}) \\ \mathrm{vec}\{\boldsymbol{\mu}_{\nabla^2 f}^{\mathrm{pref}}(\boldsymbol{x})\} \end{bmatrix}, \boldsymbol{\Sigma}_{\boldsymbol{g}}^{\mathrm{pref}}(\boldsymbol{x}) \right), \tag{9}$$

where $\boldsymbol{\Sigma}_{\boldsymbol{g}}^{\mathrm{pref}}(\boldsymbol{x})$ contains the covariance blocks between $f(\boldsymbol{x})$, $\nabla f(\boldsymbol{x})$, and $\mathrm{vec}\{\nabla^2 f(\boldsymbol{x})\}$. The induced approximate predictive means are

$$\underbrace{\mu_f^{\mathrm{pref}}(\boldsymbol{x}) = K(\boldsymbol{x}, \boldsymbol{X}) \boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}}^{-1} \hat{\boldsymbol{f}}}_{\in \mathbb{R}} \quad \underbrace{\boldsymbol{\mu}_{\nabla f}^{\mathrm{pref}}(\boldsymbol{x}) = \nabla_{\boldsymbol{x}} K(\boldsymbol{x}, \boldsymbol{X}) \boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}}^{-1} \hat{\boldsymbol{f}}}_{\in \mathbb{R}^d} \quad \underbrace{\boldsymbol{\mu}_{\nabla^2 f}^{\mathrm{pref}}(\boldsymbol{x}) = \nabla_{\boldsymbol{x}}^2 K(\boldsymbol{x}, \boldsymbol{X}) \boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}}^{-1} \hat{\boldsymbol{f}}}_{\in \mathbb{R}^{d \times d}}$$

Since $\boldsymbol{C}$ is only guaranteed to be positive semidefinite (Chu & Ghahramani, 2005), we use the regularized matrix $\hat{\boldsymbol{C}} = \boldsymbol{C} + \varepsilon_{\mathrm{reg}} \boldsymbol{I}$, where $\varepsilon_{\mathrm{reg}} > 0$. For compactness, let $\mathcal{L}_{0,\boldsymbol{x}} f = f(\boldsymbol{x})$, $\mathcal{L}_{1,\boldsymbol{x}} f = \nabla f(\boldsymbol{x})$, and $\mathcal{L}_{2,\boldsymbol{x}} f = \mathrm{vec}\{\nabla^2 f(\boldsymbol{x})\}$, and define $\boldsymbol{k}_a(\boldsymbol{x}, \boldsymbol{X}) = \mathcal{L}_{a,\boldsymbol{x}} K(\boldsymbol{x}, \boldsymbol{X})$ for $a \in \{0, 1, 2\}$. Then the covariance block between derivative orders $a, b \in \{0, 1, 2\}$ is

$$\boldsymbol{\Sigma}_{ab}^{\mathrm{pref}}(\boldsymbol{x}) = \mathcal{L}_{a,\boldsymbol{x}} \mathcal{L}_{b,\boldsymbol{x}'} k(\boldsymbol{x}, \boldsymbol{x}') \big|_{\boldsymbol{x}'=\boldsymbol{x}} - \boldsymbol{k}_a(\boldsymbol{x}, \boldsymbol{X}) \left( \boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}} + \hat{\boldsymbol{C}}^{-1} \right)^{-1} \boldsymbol{k}_b(\boldsymbol{X}, \boldsymbol{x}), \tag{10}$$

where $\boldsymbol{k}_b(\boldsymbol{X}, \boldsymbol{x}) = \boldsymbol{k}_b(\boldsymbol{x}, \boldsymbol{X})^\top$, with derivatives taken with respect to the second kernel argument. The full covariance matrix in (9) is obtained by stacking these blocks:

$$\boldsymbol{\Sigma}^{\text{pref}}_{\boldsymbol{g}}(\boldsymbol{x}) = \begin{bmatrix} \boldsymbol{\Sigma}^{\text{pref}}_{00}(\boldsymbol{x}) & \boldsymbol{\Sigma}^{\text{pref}}_{01}(\boldsymbol{x}) & \boldsymbol{\Sigma}^{\text{pref}}_{02}(\boldsymbol{x}) \\ \boldsymbol{\Sigma}^{\text{pref}}_{10}(\boldsymbol{x}) & \boldsymbol{\Sigma}^{\text{pref}}_{11}(\boldsymbol{x}) & \boldsymbol{\Sigma}^{\text{pref}}_{12}(\boldsymbol{x}) \\ \boldsymbol{\Sigma}^{\text{pref}}_{20}(\boldsymbol{x}) & \boldsymbol{\Sigma}^{\text{pref}}_{21}(\boldsymbol{x}) & \boldsymbol{\Sigma}^{\text{pref}}_{22}(\boldsymbol{x}) \end{bmatrix}. \tag{11}$$

We build our local PBO algorithms on this joint predictive formulation.

## 5 Local Search Mechanisms for Preferential Optimization

Having revisited the pairwise GP framework and derived its predictive distributions, we now address the central question of this paper: *how can we leverage local BO mechanisms to efficiently solve high-dimensional preferential optimization problems?* For this, we transfer three established local BO strategies to the preference setting. Each represents a different approach to local search in black-box optimization. For each method, we first discuss the standard BO algorithm it builds upon and then detail the adaptations required for pairwise feedback.

### 5.1 GIPBO: Gradient-based Preferential Bayesian Optimization

The first class of local BO algorithms that we reinterpret for the local PBO setting is gradient-informed search. Our algorithm GIPBO extends GIBO (Müller et al., 2021), which uses the GP surrogate to actively learn the gradients and performs first-order gradient steps, to pairwise preferential feedback.

**Gradient Information with Bayesian Optimization** GIBO (Müller et al., 2021) performs local optimization by actively reducing posterior uncertainty of the gradient at the current iterate $\boldsymbol{x}_t$. Let $\boldsymbol{\Sigma}_{11}(\boldsymbol{x}_t \mid \mathcal{D})$ denote the posterior covariance of $\nabla f(\boldsymbol{x}_t)$. At each iteration, GIBO selects $M = d$ query points using the Gradient Information acquisition function

$$\alpha_{\text{GI}}(\boldsymbol{x} \mid \boldsymbol{x}_t, \mathcal{D}) = \text{Tr}(\boldsymbol{\Sigma}_{11}(\boldsymbol{x}_t \mid \mathcal{D})) - \text{Tr}(\boldsymbol{\Sigma}_{11}(\boldsymbol{x}_t \mid \mathcal{D} \cup \{(\boldsymbol{x}, y)\})), \tag{12}$$

and then takes a step along the posterior mean gradient. In standard GP regression, the posterior covariance is independent of the observed function value $y$, so the acquisition value for a candidate $\boldsymbol{x}$ can be computed without considering possible outcomes.

**Extension to Preferential Feedback** We now extend GIBO to pairwise preferential feedback. Given two initial samples, we take the preferred one as the starting point. Each iteration repeats: *(i)* estimate the gradient at the current point via $M$ experiments; *(ii)* move one step along this estimate. We repeat these steps until the sample budget is empty. Further discussion and the full algorithm are presented in Appendix B.2 and Algorithm 1.

Naively reusing the GI acquisition function (12) for PBO fails because the Jacobian covariance of the pairwise GP $\boldsymbol{\Sigma}^{\text{pref}}_{11}$ at $\boldsymbol{x}_t$ no longer depends solely on the virtual dataset $\hat{\boldsymbol{X}}$. Instead, the pairwise GP's Jacobian variance following (10) is

$$\boldsymbol{\Sigma}^{\text{pref}}_{11} = \nabla^2_{\boldsymbol{x}_t} K(\boldsymbol{x}_t, \boldsymbol{x}_t) - \nabla_{\boldsymbol{x}_t} K(\boldsymbol{x}_t, \hat{\boldsymbol{X}}) \left( \boldsymbol{K}_{\hat{\boldsymbol{X}}\hat{\boldsymbol{X}}} + \hat{\boldsymbol{C}}^{-1} \right)^{-1} \nabla_{\boldsymbol{x}_t} K(\hat{\boldsymbol{X}}, \boldsymbol{x}_t). \tag{13}$$

where the matrix $\hat{\boldsymbol{C}}$ would now depend on the outcome of a duel $(\boldsymbol{x}_t, \boldsymbol{x})$. This forces us to consider two cases; the virtual data point being preferred over the current parameter, or vice versa; and take a conditional expectation based on the surrogate model, computed via (5). The preferential gradient-information acquisition function is therefore

$$\begin{aligned} \alpha_{\text{GI-PBO}}(\boldsymbol{x} \mid \boldsymbol{x}_t, \mathcal{D}) = \text{Tr}\left(\boldsymbol{\Sigma}^{\text{pref}}_{11}(\boldsymbol{x}_t \mid \mathcal{D})\right) &- P(\boldsymbol{x}_t \succ \boldsymbol{x}) \, \text{Tr}\left(\boldsymbol{\Sigma}^{\text{pref}}_{11}(\boldsymbol{x}_t \mid \hat{\mathcal{D}}_{\boldsymbol{x}_t \succ \boldsymbol{x}})\right) \\ &- P(\boldsymbol{x} \succ \boldsymbol{x}_t) \, \text{Tr}\left(\boldsymbol{\Sigma}^{\text{pref}}_{11}(\boldsymbol{x}_t \mid \hat{\mathcal{D}}_{\boldsymbol{x} \succ \boldsymbol{x}_t})\right) \end{aligned} \tag{14}$$

where $\hat{\mathcal{D}}_{\boldsymbol{x}_t \succ \boldsymbol{x}} := \{\mathcal{D}, ((\boldsymbol{x}_t, \boldsymbol{x}), \hat{\pi} = 0)\}$ denotes the virtual dataset for $\boldsymbol{x}_t$ being preferred over $\boldsymbol{x}$ and $\hat{\mathcal{D}}_{\boldsymbol{x} \succ \boldsymbol{x}_t}$ the opposite case. To obtain the next query $\boldsymbol{x}_{i+1}$, we maximize (14). Analogous to GIBO, we optimize the acquisition function inside search bounds around the current $\boldsymbol{x}_t$. To account for discernibility limits (Luce, 1956), we also use a lower search bound. Due to the dependence

on $\boldsymbol{C}$, we have to apply a grid-based optimization strategy by selecting points from various $d$-dimensional hyperspheres around the current parameters $\boldsymbol{x}_t$. Each virtual comparison for each of the grid points requires refitting the model, which makes the acquisition function very expensive. For high-dimensional settings, we therefore use GIPBOr, where points inside these search bounds are suggested randomly.

During gradient estimation, GIPBO maintains a local incumbent initialized at the current center point, $\boldsymbol{x}^\star_{t,0} = \boldsymbol{x}_t$. For the $i$-th local query $\boldsymbol{x}_{t,i}$, the method observes a preference between $\boldsymbol{x}_{t,i}$ and the current incumbent $\boldsymbol{x}^\star_{t,i-1}$, then updates

$$\boldsymbol{x}^\star_{t,i} = \begin{cases} \boldsymbol{x}_{t,i}, & \text{if } \boldsymbol{x}_{t,i} \succ \boldsymbol{x}^\star_{t,i-1}, \\ \boldsymbol{x}^\star_{t,i-1}, & \text{otherwise.} \end{cases} \tag{15}$$

The preferential GP is updated after each duel, and the acquisition process is repeated for $M = d$ local queries. GIPBO then estimates the local gradient direction and takes a step along $\boldsymbol{p}_t = \boldsymbol{\mu}^{\text{pref}}_{\nabla f}(\boldsymbol{x}_t)$ to the next center point $\boldsymbol{x}_{t+1}$, without performing an additional preference comparison for this step. The next iteration starts a new gradient-estimation phase centered at $\boldsymbol{x}_{t+1}$.

### 5.2 PrefSQP: Preference-based Sequential Quadratic Programming

Building on the gradient-based philosophy of GIPBO, we build PrefSQP, which additionally exploits the posterior mean Hessian $\boldsymbol{\mu}^{\text{pref}}_{\nabla^2 f}(\boldsymbol{x}_t)$ derived in Section 4. Concretely, it adapts BayeSQP (Brunzema & Trimpe, 2025) to the preference setting: scalar observations are replaced by pairwise comparisons, and a quadratic subproblem defined by the posterior mean gradient and Hessian of the pairwise GP determines the search direction. To our knowledge, PrefSQP is the first method to incorporate second-order information from a preference surrogate into the optimization step.

**Local Sub-Sampling via Pairwise Comparisons** At each iteration, BayeSQP draws $M = d$ points uniformly from a $d$-dimensional ball of radius $\varepsilon$ centered at the current iterate $\boldsymbol{x}_t$. In the standard (scalar-observation) setting, each sample produces a noisy function value. PrefSQP instead compares every sampled point $\boldsymbol{x}_i$ to the current iterate, yielding a binary preference $\pi_i \in \{0, 1\}$. All $M$ comparisons $\{(\boldsymbol{x}_t, \boldsymbol{x}_i), \pi_i\}_{i=1}^M$ are then used to update the pairwise GP.

**Subproblem Formulation** Following Brunzema & Trimpe (2025), we adopt the expected-value formulation of the BayeSQP subproblem. Since our objective is to maximize the latent utility $f$, PrefSQP seeks the local ascent direction

$$\boldsymbol{p}_t \in \underset{\boldsymbol{p} \in \mathbb{R}^d}{\arg\max} \quad \mathbb{E}\left[\frac{1}{2}\boldsymbol{p}^\top \nabla^2 f(\boldsymbol{x}_t)\boldsymbol{p} + \nabla f(\boldsymbol{x}_t)^\top \boldsymbol{p} + f(\boldsymbol{x}_t)\right] \tag{16}$$

We prefer this formulation over the uncertainty-aware variant of BayeSQP because the Laplace approximation underlying the pairwise GP already limits the fidelity of the posterior covariances, which can undermine robustness mechanisms such as the second-order cone program. Moreover, the expected-value formulation performs strongly for unconstrained objectives in the scalar setting (Brunzema & Trimpe, 2025). Since $\boldsymbol{p}$ is deterministic, the expectation reduces to the posterior quantities $\boldsymbol{H}_t \coloneqq \boldsymbol{\mu}^{\text{pref}}_{\nabla^2 f}(\boldsymbol{x}_t)$ and $\boldsymbol{\mu}^{\text{pref}}_{\nabla f}(\boldsymbol{x}_t)$. When $\boldsymbol{H}_t \prec 0$,[1] the expected quadratic problem (16) has the closed-form maximizer

$$\boldsymbol{p}_t = -\boldsymbol{H}_t^{-1}\boldsymbol{\mu}^{\text{pref}}_{\nabla f}(\boldsymbol{x}_t) = -\left[\nabla^2_{\boldsymbol{x}_t} K(\boldsymbol{x}_t, \boldsymbol{X})\boldsymbol{K}^{-1}_{\boldsymbol{X}\boldsymbol{X}}\hat{\boldsymbol{f}}\right]^{-1} \nabla_{\boldsymbol{x}_t} K(\boldsymbol{x}_t, \boldsymbol{X})\boldsymbol{K}^{-1}_{\boldsymbol{X}\boldsymbol{X}}\hat{\boldsymbol{f}}, \tag{17}$$

which is the classic Newton-type ascent direction obtained via a single linear solve.

**On Approximation Quality of the Hessian Prediction** Whether second-order information from a Laplace-approximated pairwise GP is reliable enough to be useful in a downstream algorithm is not obvious. Unlike GIPBO, where (14) actively reduces posterior gradient uncertainty at the current iterate, PrefSQP reads off the Hessian from a locally fitted pairwise GP without explicitly targeting curvature accuracy. The following identity relates the error induced by replacing the exact posterior mean with the Laplace MAP estimate to the resulting gradient and Hessian predictions.

[1] In practice, we follow Brunzema & Trimpe (2025) and enforce this condition via an eigenvalue modification.

**Lemma 1** (Laplace approximation error). *Let* $\bar{\boldsymbol{f}} := \mathbb{E}_{p(\boldsymbol{f}|\mathcal{D})}[\boldsymbol{f}]$ *denote the exact posterior mean and* $\hat{\boldsymbol{f}}$ *the MAP estimate, and define the MAP bias* $\boldsymbol{\delta} := \bar{\boldsymbol{f}} - \hat{\boldsymbol{f}}$. *Recall that* $\mathcal{L}_{1,\boldsymbol{x}} f = \nabla f(\boldsymbol{x})$, $\mathcal{L}_{2,\boldsymbol{x}} f = \mathrm{vec}\{\nabla^2 f(\boldsymbol{x})\}$, *and* $\boldsymbol{k}_a(\boldsymbol{x}, \boldsymbol{X}) = \mathcal{L}_{a,\boldsymbol{x}} K(\boldsymbol{x}, \boldsymbol{X})$. *Then, for* $k \in \{1, 2\}$,

$$\boldsymbol{\mu}^{\text{exact}}_{\mathcal{L}_k f}(\boldsymbol{x}) - \boldsymbol{\mu}^{\text{pref}}_{\mathcal{L}_k f}(\boldsymbol{x}) = \boldsymbol{k}_k(\boldsymbol{x}, \boldsymbol{X})\, \boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}}^{-1}\, \boldsymbol{\delta}. \tag{18}$$

*Proof sketch.* By the law of iterated expectations, the exact predictive mean of $\mathcal{L}_{k,\boldsymbol{x}} f$ under the true (non-Gaussian) posterior is $\boldsymbol{\mu}^{\text{exact}}_{\mathcal{L}_k f}(\boldsymbol{x}) = \boldsymbol{k}_k(\boldsymbol{x}, \boldsymbol{X})\, \boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}}^{-1}\, \bar{\boldsymbol{f}}$, while the Laplace approximation replaces $\bar{\boldsymbol{f}}$ by $\hat{\boldsymbol{f}}$, giving $\boldsymbol{\mu}^{\text{pref}}_{\mathcal{L}_k f}(\boldsymbol{x}) = \boldsymbol{k}_k(\boldsymbol{x}, \boldsymbol{X})\, \boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}}^{-1}\, \hat{\boldsymbol{f}}$. The full proof is in Appendix A. □

The identity (18) shows that the gradient and Hessian errors are driven by the same MAP bias $\boldsymbol{\delta}$, but propagated through different operators $\boldsymbol{k}_k(\boldsymbol{x}, \boldsymbol{X})\, \boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}}^{-1}$. For the Hessian ($k = 2$), this operator involves second derivatives of the kernel and can therefore be more sensitive to the kernel lengthscale, the conditioning of $\boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}}$, and the local geometry of the sampled points than the gradient case ($k = 1$). Thus, the availability of a posterior mean Hessian does not by itself imply that the resulting search direction is reliable. Whether the approximate curvature information is beneficial is therefore an empirical question; in Section 6, we find that it can be useful in practice.

**Line Search via Pairwise Comparisons** Given $\boldsymbol{p}_t$, a line search can be performed along the segment $\{\boldsymbol{x}_t + \eta\, \boldsymbol{p}_t \mid \eta \in [0, 1]\}$. In Appendix C.7, we compare several strategies including TS and EUBO (Astudillo et al., 2023) on the line as well as taking the full Newton step ($\eta = 1$). All approaches yield comparable performance in our setting with preference feedback. We therefore use TS as the line search method for PrefSQP in all experiments.

### 5.3 TuRPBO: Trust-Region Preferential Bayesian Optimization

Lastly, we construct TuRPBO (pronounced "turpebo"), as a preferential analogue of the popular TuRBO algorithm (Eriksson et al., 2019). Unlike the gradient-based methods above, TuRPBO does not incorporate explicit higher-order information into the search. Instead, it restricts the acquisition function to an adaptive trust region that expands after successful queries and contracts after failures, making it particularly straightforward to transfer to the preferential setting. Within each trust region, we use either Thompson sampling (Eriksson et al., 2019) or qEUBO (Astudillo et al., 2023) as the acquisition function.

The key transfer from TuRBO to TuRPBO is the definition of a successful step. In standard BO, a trust-region step is successful if a newly observed scalar value improves upon the best value observed so far. In TuRPBO, scalar values are unavailable; instead, a candidate $\boldsymbol{x}_{t,i}$ is considered successful if it is preferred to the current best point, i.e., if $\boldsymbol{x}_{t,i} \succ \boldsymbol{x}_t^*$. For a batch $B_t = \{\boldsymbol{x}_{t,1}, \ldots, \boldsymbol{x}_{t,q}\}$, the iteration is therefore successful if at least one candidate satisfies $\boldsymbol{x}_{t,i} \succ \boldsymbol{x}_t^*$; otherwise, $\boldsymbol{x}_t^*$ is left unchanged and the iteration is counted as a failure. The trust region is then expanded after consecutive successes and contracted after consecutive failures, exactly as in TuRBO. Full details are in Appendix B.5. This makes TuRPBO the most direct local-BO adaptation presented here: the trust-region geometry, lengthscale-based rescaling, and expansion–contraction rules remain unchanged, while the scalar improvement test is replaced by a preference-based one.

### 5.4 Warm-Starting with Global Search

Since the gradient-based local search mechanism is naturally highly dependent on its initialization and performs best on high-dimensional tasks (Wu et al., 2023), we introduce Global-to-Local (G2L) variants for GIPBO and PrefSQP. These variants use standard PBO with the EUBO acquisition function to generate the first half of the query points before switching to a local variant for the remaining queries. The global phase serves to locate a promising region before local refinement begins, with the aim of making the gradient-based local methods less dependent on their initialization and thus more robust across tasks.

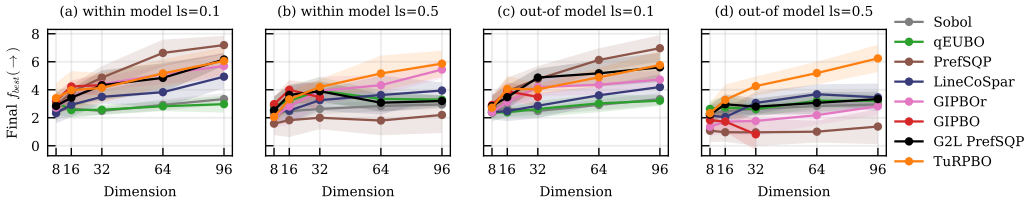



Figure 2: *Performance on GP sample paths.* We plot the final performance (mean $\pm$ std) after $10d$ iterations.

## 6 Results

We benchmark our family of local PBO methods against standard global approaches to identify when local search is beneficial. As baselines we include qEUBO (Astudillo et al., 2023), LineCoSpar (Tucker et al., 2020), and pseudo-random Sobol sampling. We report online best-seen performance, $f_{\text{best},k} = \max_{k \leq K} f_k$, as well as cumulative performance $F_k = \sum_{k=0}^{K} f_k$ over function evaluations. The latter captures the quality of the full optimization trajectory and is therefore important in controller tuning and policy search problems, where poor intermediate evaluations can be costly. Note that the algorithms only get comparisons of noisy function values (2). We repeat each experiment 10 times with a new initialization. The added noise is Gaussian with a standard deviation $10\,\%$ of the approximated function value range. All methods are evaluated with matched budgets and the same initial data. We report G2L-PrefSQP in the main text; G2L-GIPBO(r), the low-dimensional Sushi (Siivola et al., 2021) and CarCab (Lin et al., 2022) use cases, and full experimental details are in Appendix C, with ablations in Appendix C.7.

**Within-Model and Out-of-Model Comparisons** We first study the controlled setting of GP sample-path problems in dimensions 8 to 96 with squared-exponential kernels ($\ell = 0.1$ and $\ell = 0.5$), under both within-model (known hyperparameters) and out-of-model (learned hyperparameters) settings. Figure 2 reports best-seen performance after $2 + 10d$ evaluations. Local methods excel in high-dimensional, short-lengthscale problems, where global acquisition functions become inefficient; PrefSQP performs particularly well here, while TuRPBO is the most robust overall. For very smooth functions, local methods can get trapped in poor local optima and underperform global baselines. G2L-PrefSQP avoids this failure mode for $\ell = 0.5$ while remaining competitive for $\ell = 0.1$. Contrary to the derivative-based local PBO approaches, the wide initial trust region of TuRPBO allows it to escape a bad initialization and then commit to a good local region. The out-of-model comparison further reveals that qEUBO and GIPBO are sensitive to hyperparameter learning. GIPBO is evaluated only up to 32 dimensions due to computational cost.

**Evaluation on Standard Benchmarks** We next evaluate on standard benchmarks starting from two random initial points and optimizing for $10d$ iterations. Figure 3 reports best-seen and cumulative performance, confirming the main trend from the GP study: local methods become more useful as dimensionality and landscape complexity increase, achieving better cumulative performance by exploiting promising regions early rather than globally exploring the full search space. PrefSQP consistently outperforms GIPBO, indicating that second-order posterior information is more effective for local refinement than estimating gradients alone despite the Laplace-approximate GP posterior, and is mostly competitive with TuRPBO, outperforming it in the highest-dimensional setting. The main exception is Hartmann, where TuRPBO performs best, consistent with the results on smoother objectives from Sec. 6, while G2L-PrefSQP remains competitive. We report wall-clock time: Sobol, LineCoSpar and GIPBOr are inexpensive, while GIPBO becomes costly as dimension increases. qEUBO, (G2L-)PrefSQP and TuRPBO remain feasible also in high dimensions. Details are reported in Table 12. It also highlights a trade-off: derivative-informed PBO can improve sample efficiency, but not all derivative-informed methods scale equally well computationally.

**Policy Search** Lastly, we evaluate the baselines on MuJoCo Hopper (33-$d$) and Walker2D (102-$d$) policy-search tasks, using linear feedback policies following (Mania et al., 2018), where each iteration is one full episode rollout. Each run is warm-started with $5d$ random evaluations; the best observed policy initializes the preference model, and each method runs for $10d$ iterations, simulating a real-world setting with a crude initial policy. Figure 4 shows that TuRPBO achieves the strongest final

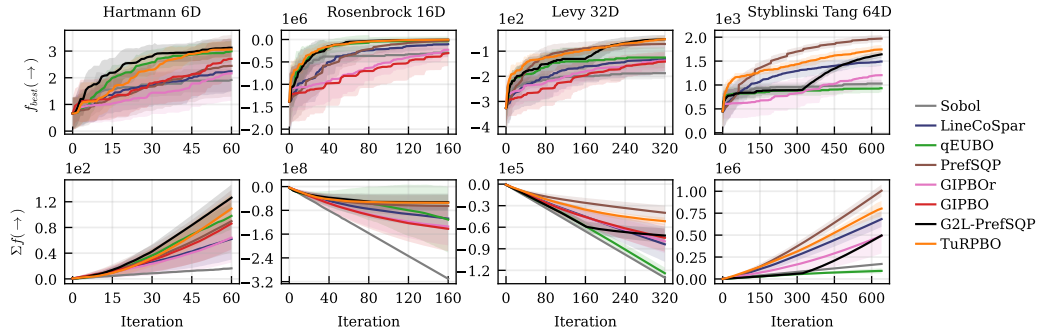



Figure 3: *Performance on synthetic benchmarks.* Best (top) and cumulative (bottom) performance (mean ± std). Our local PBO methods achieve higher cumulative performance in high-dimensional settings.

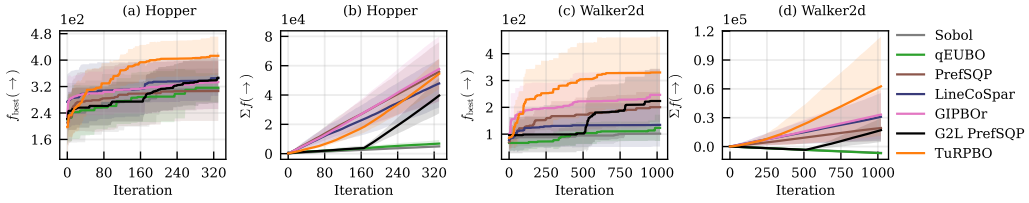



Figure 4: *Performance on policy-search tasks Hopper and Walker2D.*

performance in both tasks. Overall, our local PBO methods achieve higher cumulative performance than global baselines by avoiding low-return evaluations, making them particularly useful for high-dimensional policy tuning with a reasonable initial region.

## 7 Discussion on Limitations and Future Work

While our results show that local search can improve PBO in high-dimensional settings, some limitations remain. First, we studied the proposed local mechanisms separately and evaluated global warmstarting using a fixed 50% budget split. Although this already improves performance, the optimal balance between global and local search likely depends on the problem. Future work should therefore investigate adaptive warmstarting strategies that vary the acquisition function and switching point. Second, apart from the low-dimensional Sushi and CarCab benchmarks, our evaluations build on synthetic objectives and policy-search problems. Although these benchmarks are useful for controlled comparisons, they do not fully capture the structure of human preferences. In real applications (Coutinho et al., 2024; Erarslan et al., 2025; Dewancker et al., 2018), experiments and preferences may be noisier, inconsistent, context-dependent, or limited by a threshold of discernibility, where two alternatives are effectively indistinguishable to the user. Further, some real-world problems are prone to experimental "crashes" (Menn et al., 2026), motivating a more cautious exploration through local search. Future work should evaluate local PBO on real systems with human feedback on high-dimensional problems. Finally, the effect of alternative comparison mechanisms, such as larger comparison batches (Astudillo et al., 2023), ranking (Nguyen et al., 2021), or explicit indifference responses (Bemporad & Piga, 2021; Erarslan et al., 2025) on user burden and robustness remains an interesting direction for future work.

## 8 Conclusion

This work presented a family of local PBO methods that adapt trust-region optimization and derivative-informed refinement to pairwise preference feedback. Our experiments show that restricting search to promising regions addresses a central scalability challenge of PBO, yielding clear gains in high-dimensional, complex landscapes. Among the proposed methods, TuRPBO offers the most robust performance, while PrefSQP demonstrates that second-order posterior information from a Laplace-approximated pairwise GP can meaningfully improve local refinement. Global-to-local warm-starting further reduces the dependence of gradient-based local methods on a good initialization. These findings position local PBO as well suited for applications such as controller tuning and policy search, where costly intermediate evaluations should be avoided.

### AI USE STATEMENT

In this work, we used generative AI tools for code generation and text revision. In particular, generative AI tools assisted with implementing and debugging code used to run experiments and produce the reported experimental results. We have not used generative AI tools to directly generate or fabricate experimental results, data, or scientific claims, and the remaining required disclosure tasks are not applicable to this work. Additionally, we used generative AI tools to improve the wording, clarity, and readability of the manuscript. We have reviewed all AI-assisted work. AI-generated code was manually reviewed and tested for correctness, and all AI-assisted manuscript revisions were checked by the authors for accuracy and consistency with the intended scientific claims. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

### ETHICS STATEMENT

This work studies preference-based optimization as a general methodological problem. Our experiments are conducted exclusively on synthetic benchmark functions and artificial benchmark environments and do not involve human subjects, personal data, or real-world decision-making. As a general optimization method, the proposed approach could in principle be applied to a wide range of domains, including applications with societal or ethical implications. Such applications are outside the scope of this work and would require domain-specific consideration of safety, fairness, privacy, and other relevant ethical requirements. We are not aware of additional ethical concerns arising from the experiments presented in this paper.

### REPRODUCIBILITY STATEMENT

We describe the main algorithms in Section 5 and provide full pseudocode in Appendix B. Appendix C contains details on the test functions, surrogate models, and algorithm configurations. All experiments were implemented in Python and build on the open-source BoTorch package (Balandat et al., 2020). Additional software dependencies are listed in Table 17.

## REFERENCES

Raul Astudillo, Zhiyuan Lin, Eytan Bakshy, and Peter I. Frazier. qEUBO: A decision-theoretic acquisition function for preferential Bayesian optimization. In *International Conference on Artificial Intelligence and Statistics (AISTATS), 25-27 April 2023, Palau de Congressos, Valencia, Spain*, volume 206 of *Proceedings of Machine Learning Research*, pp. 1093–1114. PMLR, 2023.

Maximilian Balandat, Brian Karrer, Daniel R. Jiang, Samuel Daulton, Benjamin Letham, Andrew Gordon Wilson, and Eytan Bakshy. BoTorch: A framework for efficient Monte-Carlo Bayesian optimization. In *Advances in Neural Information Processing Systems (NeurIPS) 33*, 2020.

Alberto Bemporad. Global optimization via inverse distance weighting and radial basis functions. *Computational Optimization and Applications*, 77(2):571–595, 2020.

Alberto Bemporad and Dario Piga. Global optimization based on active preference learning with radial basis functions. *Machine Learning*, 110(2):417–448, 2021.

Alessio Benavoli, Dario Azzimonti, and Dario Piga. Preferential Bayesian optimisation with skew Gaussian processes. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 1842–1850, 2021.

Eric Brochu, Vlad M Cora, and Nando De Freitas. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*, 2010.

Paul Brunzema and Sebastian Trimpe. BayeSQP: Bayesian optimization through sequential quadratic programming. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Wei Chu and Zoubin Ghahramani. Preference learning with Gaussian processes. In *Proceedings of the 22nd International Conference on Machine Learning (ICML)*, pp. 137–144, 2005.

Joao PL Coutinho, Ivan Castillo, and Marco S Reis. Human-in-the-loop controller tuning using preferential Bayesian optimization. *IFAC-PapersOnLine*, 58(14):13–18, 2024.

Stefano De Blasi, Maryam Bahrami, Elmar Engels, and Alexander Gepperth. Safe contextual Bayesian optimization integrated in industrial control for self-learning machines. *Journal of Intelligent Manufacturing*, 35:885–903, 2024. doi: 10.1007/s10845-023-02087-3.

Ian Dewancker, Jakob Bauer, and Michael McCourt. Sequential preference-based optimization. *arXiv preprint arXiv:1801.02788*, 2018.

Aras Erarslan, Carlos Sevilla Salcedo, Ville Tanskanen, Anni Nisov, Eero Päiväkumpu, Heikki Aisala, Kaisu Honkapää, Arto Klami, and Petrus Mikkola. Consecutive preferential Bayesian optimization, November 2025. arXiv:2511.05163 [cs].

David Eriksson and Matthias Poloczek. Scalable constrained Bayesian optimization. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2021.

David Eriksson, Michael Pearce, Jacob Gardner, Ryan D Turner, and Matthias Poloczek. Scalable global optimization via local Bayesian optimization. *Advances in Neural Information Processing Systems (NeurIPS)*, 32, 2019.

Zheyi Fan, Wenyu Wang, Szu H Ng, and Qingpei Hu. Minimizing UCB: a better local search strategy in local Bayesian optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

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

Javier González, Zhenwen Dai, Andreas Damianou, and Neil D Lawrence. Preferential Bayesian optimization. In *International Conference on Machine Learning (ICML)*, pp. 1282–1291. PMLR, 2017.

Shiming He, Alexander von Rohr, Dominik Baumann, Ji Xiang, and Sebastian Trimpe. Simulation-aided policy tuning for black-box robot learning. *IEEE Transactions on Robotics (TRO)*, 2025.

Antonia Holzapfel, Paul Brunzema, and Sebastian Trimpe. Event-triggered safe Bayesian optimization on quadcopters. In *6th Annual Learning for Dynamics & Control Conference*, pp. 1033–1045. PMLR, 2024.

Henrik Hose, Paul Brunzema, Alexander von Rohr, Alexander Gräfe, Angela P. Schoellig, and Sebastian Trimpe. Fine-tuning of neural network approximate MPC without retraining via Bayesian optimization. *International Conference on Robot Intelligence Technologies and Applications (RiTA)*, 2025.

Joshua Hang Sai Ip, Ankush Chakrabarty, Ali Mesbah, and Diego Romeres. User preference meets pareto-optimality in multi-objective Bayesian optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 20246–20254, 2025.

Johannes Kirschner, Mojmir Mutny, Nicole Hiller, Rasmus Ischebeck, and Andreas Krause. Adaptive and safe Bayesian optimization in high dimensions via one-dimensional subspaces. In *International Conference on Machine Learning (ICML)*, pp. 3429–3438. PMLR, 2019.

Satoshi Koide and Ayano Okoso. High-dimensional slider-based preferential Bayesian optimization with mixed local and global acquisition strategies. *IEEE Access*, 2024.

Zhiyuan Jerry Lin, Raul Astudillo, Peter Frazier, and Eytan Bakshy. Preference exploration for efficient Bayesian optimization with multiple outcomes. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 4235–4258. PMLR, 2022.

R. Duncan Luce. Semiorders and a theory of utility discrimination. *Econometrica*, 24(2):178–191, 1956.

Horia Mania, Aurelia Guy, and Benjamin Recht. Simple random search of static linear policies is competitive for reinforcement learning. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems (NeurIPS)*, volume 31. Curran Associates, Inc., 2018.

George Marsaglia. Choosing a point from the surface of a sphere. *The Annals of Mathematical Statistics*, pp. 645–646, 1972.

Johanna Menn, Pietro Pelizzari, Michael Fleps-Dezasse, and Sebastian Trimpe. Lipschitz safe Bayesian optimization for automotive control. In *2024 IEEE 63rd Conference on Decision and Control (CDC)*, pp. 3782–3788, 2024. doi: 10.1109/CDC56724.2024.10886680.

Johanna Menn, David Stenger, and Sebastian Trimpe. Preferential Bayesian optimization with crash feedback. *IEEE Robotics and Automation Letters*, 11(4):4299–4306, 2026. doi: 10.1109/LRA.2026.3665446.

Petrus Mikkola, Milica Todorović, Jari Järvi, Patrick Rinke, and Samuel Kaski. Projective preferential Bayesian optimization. In *International Conference on Machine Learning (ICML)*, pp. 6884–6892. PMLR, 2020.

Sarah Müller, Alexander von Rohr, and Sebastian Trimpe. Local policy search with Bayesian optimization. *Advances in Neural Information Processing Systems (NeurIPS)*, 34:20708–20720, 2021.

Matthias Neumann-Brosig, Alonso Marco, Dieter Schwarzmann, and Sebastian Trimpe. Data-efficient autotuning with Bayesian optimization: An industrial control study. *IEEE Transactions on Control Systems Technology*, 28(3):730–740, 2019.

Quan Nguyen, Kaiwen Wu, Jacob Gardner, and Roman Garnett. Local Bayesian optimization via maximizing probability of descent. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, 2022.

Quoc Phong Nguyen, Sebastian Tay, Bryan Kian Hsiang Low, and Patrick Jaillet. Top-k ranking Bayesian optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 9135–9143, 2021.

Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian processes for machine learning*. Adaptive computation and machine learning. MIT Press, 2006. ISBN 026218253X.

Eero Siivola, Akash Kumar Dhaka, Michael Riis Andersen, Javier González, Pablo García Moreno, and Aki Vehtari. Preferential batch bayesian optimization. In *2021 IEEE 31st International Workshop on Machine Learning for Signal Processing (MLSP)*, pp. 1–6. IEEE, 2021.

Shion Takeno, Masahiro Nomura, and Masayuki Karasuyama. Towards practical preferential Bayesian optimization with skew Gaussian processes. In *International Conference on Machine Learning (ICML)*, pp. 33516–33533. PMLR, 2023.

Wei-Ting Tang and Joel A. Paulson. CAGES: Cost-aware gradient entropy search for efficient local multi-fidelity Bayesian optimization. In *2024 IEEE 63rd Conference on Decision and Control (CDC)*, pp. 1547–1552, December 2024. doi: 10.1109/CDC56724.2024.10886516. ISSN: 2576-2370.

Wei-Ting Tang, Akshay Kudva, and Joel Paulson. NeST-BO: Fast local Bayesian optimization via Newton-step targeting of gradient and Hessian information. In *The 29th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2026.

Maegan Tucker, Myra Cheng, Ellen Novoseller, Richard Cheng, Yisong Yue, Joel W. Burdick, and Aaron D. Ames. Human preference-based learning for high-dimensional optimization of exoskeleton walking gaits. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3423–3430, 2020. doi: 10.1109/IROS45743.2020.9341416.

Maegan Tucker, Kejun Li, Yisong Yue, and Aaron D Ames. Polar: Preference optimization and learning algorithms for robotics. *arXiv preprint arXiv:2208.04404*, 2022.

Alexander von Rohr, David Stenger, Dominik Scheurenberg, and Sebastian Trimpe. Local Bayesian optimization for controller tuning with crash constraints. *at-Automatisierungstechnik*, 2024.

Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945. ISSN 00994987.

Kaiwen Wu, Kyurae Kim, Roman Garnett, and Jacob Gardner. The behavior and convergence of local Bayesian optimization. *Advances in Neural Information Processing Systems (NeurIPS)*, 36: 73497–73523, 2023.

Wenjie Xu, Wenbin Wang, Yuning Jiang, Bratislav Svetozarevic, and Colin Jones. Principled preferential Bayesian optimization. In *International Conference on Machine Learning (ICML)*, pp. 55305–55336. PMLR, 2024.

Zeji Yi, Yunyue Wei, Chu Xin Cheng, Kaibo He, and Yanan Sui. Improving sample efficiency of high dimensional Bayesian optimization with MCMC. In *6th Annual Learning for Dynamics & Control Conference*, pp. 813–824. PMLR, 2024.

Yangwenhui Zhang, Hong Qian, Xiang Shu, and Aimin Zhou. High-dimensional dueling optimization with preference embedding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 11280–11288, 2023.

Max Gero Zimmermann, Johanna Menn, Christopher Ullmann, Viktor Glushych, Thomas Schopphoven, Wilhelm Meiners, Sebastian Trimpe, and Carlo Holly. Bayesian optimization for extreme high-speed laser material deposition. *Journal of Laser Applications*, 37(4), 2025.

## A Proof of Lemma 1

We restate the lemma and provide the full proof. Recall the notation from Section 4: $\mathcal{L}_{1,\boldsymbol{x}} f = \nabla f(\boldsymbol{x})$, $\mathcal{L}_{2,\boldsymbol{x}} f = \mathrm{vec}\{\nabla^2 f(\boldsymbol{x})\}$, and $\boldsymbol{k}_a(\boldsymbol{x}, \boldsymbol{X}) = \mathcal{L}_{a,\boldsymbol{x}} K(\boldsymbol{x}, \boldsymbol{X})$.

**Assumption 1.** *The covariance function $k(\boldsymbol{x}, \boldsymbol{x}')$ is at least four times differentiable in its first argument.*

**Assumption 2.** *The prior covariance matrix $\boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}} = K(\boldsymbol{X}, \boldsymbol{X})$ is strictly positive definite.*

**Assumption 3.** *The exact posterior $p(\boldsymbol{f} \mid \mathcal{D})$ satisfies $\mathbb{E}_{p(\boldsymbol{f}|\mathcal{D})}[\|\boldsymbol{f}\|_2] < \infty$, so that the posterior mean $\bar{\boldsymbol{f}} := \mathbb{E}_{p(\boldsymbol{f}|\mathcal{D})}[\boldsymbol{f}]$ is well defined and the interchange of differentiation and integration below is justified.*

**Lemma** (Laplace approximation error for gradient and Hessian predictions, restated)**.** *Under Assumptions 1–3, let $\bar{\boldsymbol{f}} := \mathbb{E}_{p(\boldsymbol{f}|\mathcal{D})}[\boldsymbol{f}]$ denote the exact posterior mean and $\hat{\boldsymbol{f}}$ the MAP estimate, and define the MAP bias $\boldsymbol{\delta} := \bar{\boldsymbol{f}} - \hat{\boldsymbol{f}}$. Then, for any test point $\boldsymbol{x} \in \Theta$ and $k \in \{1, 2\}$,*

$$\boldsymbol{\mu}^{\mathrm{exact}}_{\mathcal{L}_k f}(\boldsymbol{x}) - \boldsymbol{\mu}^{\mathrm{pref}}_{\mathcal{L}_k f}(\boldsymbol{x}) = \boldsymbol{k}_k(\boldsymbol{x}, \boldsymbol{X})\, \boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}}^{-1}\, \boldsymbol{\delta}. \tag{18}$$

*Proof.* The proof proceeds in two steps: we first establish the predictive mean under an arbitrary posterior and then pass to derivatives to obtain the exact identity (18).

For a GP with prior mean zero and prior covariance $k$, the conditional distribution of the function value $f(\boldsymbol{x})$ at a test point $\boldsymbol{x}$, given the latent vector $\boldsymbol{f}$ at the training locations $\boldsymbol{X}$, is

$$p(f(\boldsymbol{x}) \mid \boldsymbol{f}) = \mathcal{N}\big(K(\boldsymbol{x}, \boldsymbol{X})\, \boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}}^{-1}\, \boldsymbol{f},\ k(\boldsymbol{x}, \boldsymbol{x}) - K(\boldsymbol{x}, \boldsymbol{X})\, \boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}}^{-1}\, K(\boldsymbol{X}, \boldsymbol{x})\big). \tag{19}$$

This conditional is a property of the GP prior and holds regardless of the form of the posterior $p(\boldsymbol{f} \mid \mathcal{D})$. Taking the predictive expectation under *any* posterior $p(\boldsymbol{f} \mid \mathcal{D})$:

$$\begin{aligned}
\mathbb{E}[f(\boldsymbol{x}) \mid \mathcal{D}] &= \int f(\boldsymbol{x})\, p(f(\boldsymbol{x}) \mid \boldsymbol{f})\, p(\boldsymbol{f} \mid \mathcal{D})\, d\boldsymbol{f} \\
&= \int K(\boldsymbol{x}, \boldsymbol{X})\, \boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}}^{-1}\, \boldsymbol{f}\ \ p(\boldsymbol{f} \mid \mathcal{D})\, d\boldsymbol{f} \\
&= K(\boldsymbol{x}, \boldsymbol{X})\, \boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}}^{-1}\, \mathbb{E}_{p(\boldsymbol{f}|\mathcal{D})}[\boldsymbol{f}].
\end{aligned} \tag{20}$$

In particular, the exact predictive mean is

$$\mu_f^{\mathrm{exact}}(\boldsymbol{x}) = K(\boldsymbol{x}, \boldsymbol{X})\, \boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}}^{-1}\, \bar{\boldsymbol{f}}, \tag{21}$$

and the Laplace-approximate predictive mean is

$$\mu_f^{\mathrm{pref}}(\boldsymbol{x}) = K(\boldsymbol{x}, \boldsymbol{X})\, \boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}}^{-1}\, \hat{\boldsymbol{f}}. \tag{22}$$

Under Assumptions 1 and 3, the linear differential operators $\mathcal{L}_{k,\boldsymbol{x}}$ can be interchanged with the posterior expectation. For $k = 1$ (gradient, $\mathcal{L}_{1,\boldsymbol{x}} f = \nabla f(\boldsymbol{x})$):

$$\boldsymbol{\mu}^{\mathrm{exact}}_{\mathcal{L}_1 f}(\boldsymbol{x}) = \mathcal{L}_{1,\boldsymbol{x}}\, \mu_f^{\mathrm{exact}}(\boldsymbol{x}) = \boldsymbol{k}_1(\boldsymbol{x}, \boldsymbol{X})\, \boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}}^{-1}\, \bar{\boldsymbol{f}}, \tag{23}$$

$$\boldsymbol{\mu}^{\mathrm{pref}}_{\mathcal{L}_1 f}(\boldsymbol{x}) = \mathcal{L}_{1,\boldsymbol{x}}\, \mu_f^{\mathrm{pref}}(\boldsymbol{x}) = \boldsymbol{k}_1(\boldsymbol{x}, \boldsymbol{X})\, \boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}}^{-1}\, \hat{\boldsymbol{f}}. \tag{24}$$

For $k = 2$ (vectorized Hessian, $\mathcal{L}_{2,\boldsymbol{x}} f = \mathrm{vec}\{\nabla^2 f(\boldsymbol{x})\}$):

$$\boldsymbol{\mu}^{\mathrm{exact}}_{\mathcal{L}_2 f}(\boldsymbol{x}) = \boldsymbol{k}_2(\boldsymbol{x}, \boldsymbol{X})\, \boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}}^{-1}\, \bar{\boldsymbol{f}}, \tag{25}$$

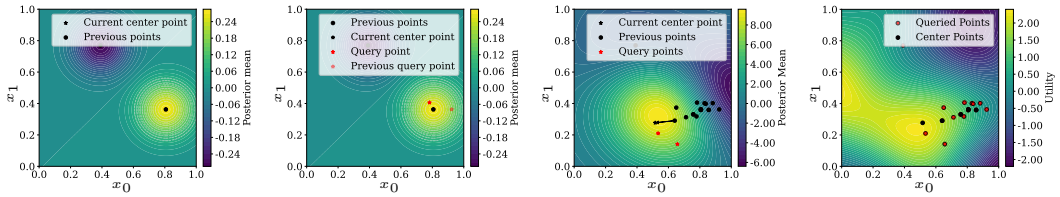
$$\boldsymbol{\mu}^{\mathrm{pref}}_{\mathcal{L}_2 f}(\boldsymbol{x}) = \boldsymbol{k}_2(\boldsymbol{x}, \boldsymbol{X})\, \boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}}^{-1}\, \hat{\boldsymbol{f}}. \tag{26}$$

Subtracting (24) from (23) (respectively (26) from (25)) yields, for $k \in \{1, 2\}$,

$$\boldsymbol{\mu}^{\mathrm{exact}}_{\mathcal{L}_k f}(\boldsymbol{x}) - \boldsymbol{\mu}^{\mathrm{pref}}_{\mathcal{L}_k f}(\boldsymbol{x}) = \boldsymbol{k}_k(\boldsymbol{x}, \boldsymbol{X})\, \boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}}^{-1}\, (\bar{\boldsymbol{f}} - \hat{\boldsymbol{f}}) = \boldsymbol{k}_k(\boldsymbol{x}, \boldsymbol{X})\, \boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}}^{-1}\, \boldsymbol{\delta}, \tag{27}$$

which is the exact identity (18).

□




(a) Initial GP posterior (b) Gradient estimation (c) Gradient step (d) Result on true function

Figure 5: Illustration of gradient steps in GIPBO. (a) The GP posterior is initialized with two data points. (b) A gradient estimate is obtained by querying $d$ samples (red query points) computed by the adapted GI acquisition function. (c) A gradient step is performed. (d) The result of five gradient steps is visualized on the true function's plot. As we do not observe function measurements but only comparisons, the output scale of the GP gets rescaled in every iteration.

# B Implementation Details and Full Algorithms

## B.1 Overview of Implementation Choices

All experiments are performed on normalized domains $\Theta = [0, 1]^d$, and all algorithms receive only noisy pairwise comparisons rather than direct function values. The local methods use the Laplace-approximated pairwise GP surrogate introduced in Section 4 to model the latent utility from these comparisons. The following sections provide implementation details that are omitted from the main text, including acquisition optimization, local sampling strategies, step normalization, trust-region updates, and the full algorithmic procedures used in the experiments.

## B.2 GIPBO: Implementation Details

### B.2.1 Algorithmic overview

At a high level, one GIPBO iteration consists of a local gradient-estimation phase followed by a gradient step to update the center point. Figure 5 illustrates the algorithm and the full algorithm is described in Algorithm 1. The algorithm starts with one comparison (Figure 5a). The winner of this comparison is then selected as the current center point $x_t$. Starting from this point, GIPBO sequentially selects $M = d$ local query points using the preferential GI acquisition function (Figure 5b). Each query is compared against the current local incumbent, the pairwise GP is refit, and the next query is selected under the updated posterior. After the $d$ comparisons, the posterior mean gradient at $x_t$ defines a normalized ascent direction, and the next center point is obtained by a projected fixed-length step (Figure 5c). The step itself is not queried immediately; it becomes the center of the next gradient-estimation phase.

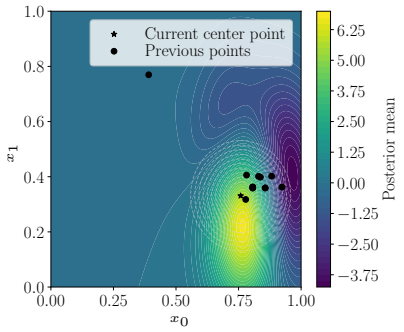
### B.2.2 Preferential GI acquisition evaluation

In contrast to GIBO, the matrix $(\boldsymbol{K}_{\hat{\boldsymbol{X}}\hat{\boldsymbol{X}}} + \hat{\boldsymbol{C}}^{-1})^{-1}$ in the preferential Jacobian covariance has to be recomputed separately for both possible virtual preference outcomes at each candidate $x$. Since $\hat{C}$ depends on the outcome of the duel, the efficient Cholesky updates used in GIBO are not directly applicable. For each virtual dataset, we therefore compute $(\boldsymbol{K}_{\hat{\boldsymbol{X}}\hat{\boldsymbol{X}}} + \hat{\boldsymbol{C}}^{-1})^{-1}$ by a full Cholesky decomposition when evaluating $\boldsymbol{\Sigma}_{11}^{\text{pref}}$.

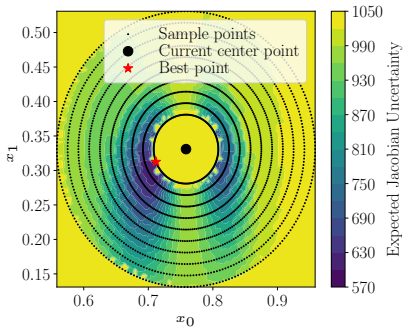
### B.2.3 Candidate-set construction

For the acquisition function optimization, we sample virtual candidates from multiple $d$-dimensional hyperspheres centered at $\boldsymbol{x}_t$. To respect discernibility limits, we enforce a minimum radius of $0.05$. In two dimensions, we draw 2,500 candidates from 10 concentric circles with radii spanning the local search bounds. Figure 6 illustrates this sampling procedure. For $d > 2$, we sample $d \times r$ candidates, where the resolution is fixed to $r = 50$. To reduce the cost compared with uniform sampling over the full local search region, we restrict the candidates to four hyperspheres with radii $0.05, 0.1, 0.15$, and $0.2$. Points on each hypersphere are generated using the Marsaglia method Marsaglia (1972).

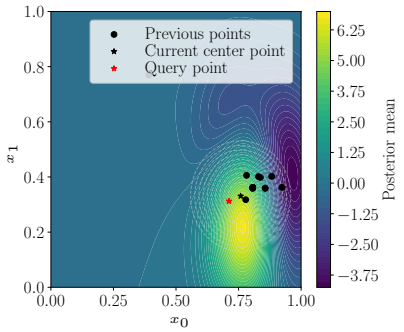
In high-dimensional experiments, we cap the number of evaluated acquisition candidates at 512 for computational reasons. This cap makes full GIPBO less reliable in high dimensions because the capped candidate set can become too sparse to approximate the acquisition maximizer accurately.



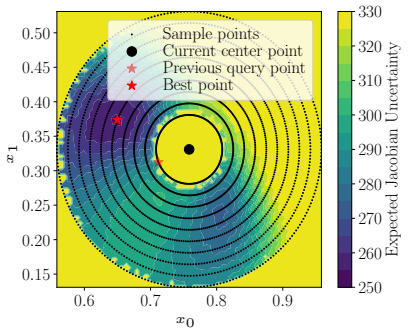

(a) GP posterior mean after the first three gradient steps. The search area for the acquisition function is displayed in light grey.




(b) Computation of the first query point to minimize the expected Jacobian uncertainty at the current parameters, here referred to as center point.




(c) Result of GP update with the duel of the center and first query point.




(d) Computation of the second query point needed for the gradient estimation.

Figure 6: GP posterior mean for parameters $\boldsymbol{x} = \{x_0, x_1\}$ and corresponding computation of the two query points to perform the fourth gradient step.

### B.2.4 Gradient-step normalization

We use a squared-exponential kernel on the normalized domain $\Theta = [0,1]^d$, with lengthscales constrained to $[0.05, 0.5]$. This constraint avoids excessively large local steps, since the gradient direction is normalized with respect to the squared-exponential lengthscales, following Müller et al. (2021). We set the minimum absolute step length to $0.05$ and rescale steps whenever they would fall below this threshold. Given the posterior mean gradient at the current iterate $\boldsymbol{x}_t$,

$$\boldsymbol{g}_t = \boldsymbol{\mu}^{\text{pref}}_{\nabla f}(\boldsymbol{x}_t),$$

we use the Mahalanobis norm induced by the squared-exponential lengthscale matrix $L$,

$$\boldsymbol{p}_t = \frac{\boldsymbol{g}_t}{\|\boldsymbol{g}_t\|_L}, \qquad \|\boldsymbol{g}_t\|_L = \sqrt{\boldsymbol{g}_t^\top L \boldsymbol{g}_t},$$

and update the iterate according to $\boldsymbol{x}_{t+1} = \Pi_\Theta(\boldsymbol{x}_t + \eta \boldsymbol{p}_t)$.

## B.3 GIPBOr: Randomized GIPBO Variant

GIPBOr uses the same pairwise GP model, local incumbent update, and normalized gradient step as GIPBO, but replaces the expensive maximization of the preferential GI acquisition function by

random local candidate selection. In each gradient-estimation phase, query points are sampled from the same local search region used by GIPBO. This variant isolates the effect of the gradient-based local update from the cost of acquisition optimization and provides a scalable approximation in higher-dimensional settings, where evaluating both virtual preference outcomes for many candidates becomes prohibitively expensive.

#### B.3.1 FULL ALGORITHM

**Algorithm 1** GIPBO

Initialize data set $\mathcal{D}_0$, current iterate $\boldsymbol{x}_0$, step length $\eta$, acquisition search bounds $(\delta_{\min}, \delta_{\max})$
**for** $t = 0, \dots, T-1$ **do**
  Set local incumbent $\boldsymbol{x}^\star_{t,0} \leftarrow \boldsymbol{x}_t$
  **for** $i = 1, \dots, M$ **do**
    Fit pairwise GP model using $\mathcal{D}_t$
    Select local query

$$\boldsymbol{x}_{t,i} \in \arg\max_{\boldsymbol{x} \in \mathcal{B}(\boldsymbol{x}_t; \delta_{\min}, \delta_{\max}) \cap \Theta} \alpha_{\text{GI-PBO}}(\boldsymbol{x} \mid \boldsymbol{x}_t, \mathcal{D}_t)$$

    Compare local query $\boldsymbol{x}_{t,i}$ against local incumbent $\boldsymbol{x}^\star_{t,i-1}$
    Observe ordered comparison $(\boldsymbol{x}^+_{t,i}, \boldsymbol{x}^-_{t,i})$
    Update $\mathcal{D}_t \leftarrow \mathcal{D}_t \cup \{(\boldsymbol{x}^+_{t,i}, \boldsymbol{x}^-_{t,i})\}$
    **if** $\boldsymbol{x}_{t,i}$ is preferred to $\boldsymbol{x}^\star_{t,i-1}$ **then**
      $\boldsymbol{x}^\star_{t,i} \leftarrow \boldsymbol{x}_{t,i}$
    **else**
      $\boldsymbol{x}^\star_{t,i} \leftarrow \boldsymbol{x}^\star_{t,i-1}$
    **end if**
  **end for**
  Fit pairwise GP model using $\mathcal{D}_t$
  Compute posterior mean gradient $\boldsymbol{g}_t \leftarrow \boldsymbol{\mu}^{\text{pref}}_{\nabla f}(\boldsymbol{x}_t)$
  Normalize gradient direction $\boldsymbol{p}_t \leftarrow \boldsymbol{g}_t / \|\boldsymbol{g}_t\|_L$
  Update iterate $\boldsymbol{x}_{t+1} \leftarrow \Pi_\Theta(\boldsymbol{x}_t + \eta \boldsymbol{p}_t)$
  Set $\mathcal{D}_{t+1} \leftarrow \mathcal{D}_t$
**end for**
**return** best observed incumbent $\boldsymbol{x}^\star$

### B.4 PREFSQP: IMPLEMENTATION DETAILS

PrefSQP follows the same preference-modeling setup as GIPBO but uses both the posterior mean gradient and posterior mean Hessian of the Laplace-approximated pairwise GP. At each iteration, $d$ local points are sampled in a ball of radius $\epsilon$ around the current iterate and compared to the current iterate. The resulting comparisons are added to the dataset and the pairwise GP is refit.

Given the posterior mean gradient $\boldsymbol{g}_t = \boldsymbol{\mu}^{\text{pref}}_{\nabla f}(\boldsymbol{x}_t)$ and Hessian $\boldsymbol{H}_t = \boldsymbol{\mu}^{\text{pref}}_{\nabla^2 f}(\boldsymbol{x}_t)$, PrefSQP constructs a local quadratic model. Since the objective is maximized, we modify the Hessian eigenvalues to enforce negative definiteness before computing the Newton-type direction. The resulting direction is used in a one-dimensional preference-based line search over $\eta \in [0, 1]$. In the experiments, this line search is implemented using Thompson sampling over a fixed grid of 100 candidate step sizes.

At each iteration, we sample $M = d$ local points from $B(\boldsymbol{x}_t, \epsilon) \cap \Theta$ and compare each of them against the current iterate $\boldsymbol{x}_t$. The resulting pairwise observations are added to the data set, and the pairwise GP is refit.

After the local comparisons, we compute

$$\boldsymbol{g}_t = \boldsymbol{\mu}^{\text{pref}}_{\nabla f}(\boldsymbol{x}_t), \qquad \boldsymbol{H}_t = \boldsymbol{\mu}^{\text{pref}}_{\nabla^2 f}(\boldsymbol{x}_t).$$

These quantities define a local quadratic model. Since the latent utility is maximized, we require a locally concave quadratic approximation before computing the Newton-type ascent direction. The

**Algorithm 2** PrefSQP

Initialize data set $\mathcal{D}_0$, current iterate $\boldsymbol{x}_0$, local sampling radius $\epsilon$, line-search resolution $R$
**for** $t = 0, \dots, T-1$ **do**
  Draw local samples $\{\boldsymbol{x}_{t,1}, \dots, \boldsymbol{x}_{t,M}\} \subset \mathcal{B}_\epsilon(\boldsymbol{x}_t) \cap \Theta$, with $M = d$
  **for** $i = 1, \dots, M$ **do**
    Compare local sample $\boldsymbol{x}_{t,i}$ against current iterate $\boldsymbol{x}_t$
    Observe ordered comparison $(\boldsymbol{x}_{t,i}^+, \boldsymbol{x}_{t,i}^-)$
    Update $\mathcal{D}_t \leftarrow \mathcal{D}_t \cup \{(\boldsymbol{x}_{t,i}^+, \boldsymbol{x}_{t,i}^-)\}$
  **end for**
  Fit pairwise GP model using $\mathcal{D}_t$
  Compute posterior mean gradient $\boldsymbol{g}_t \leftarrow \boldsymbol{\mu}_{\nabla f}^{\mathrm{pref}}(\boldsymbol{x}_t)$
  Compute posterior mean Hessian $\boldsymbol{H}_t \leftarrow \boldsymbol{\mu}_{\nabla^2 f}^{\mathrm{pref}}(\boldsymbol{x}_t)$
  Modify $\boldsymbol{H}_t$ to obtain a negative definite matrix $\widetilde{\boldsymbol{H}}_t$
  Compute SQP direction $\boldsymbol{p}_t \leftarrow -\widetilde{\boldsymbol{H}}_t^{-1}\boldsymbol{g}_t$
  Construct line-search set

$$\mathcal{L}_t = \left\{ \Pi_\Theta(\boldsymbol{x}_t + \eta_r \boldsymbol{p}_t) \ : \ \eta_r \in \left\{ 0, \frac{1}{R-1}, \dots, 1 \right\} \right\}$$

  Select $\boldsymbol{x}_{t+1} \in \mathcal{L}_t$ using Thompson sampling under the pairwise GP
  Set $\mathcal{D}_{t+1} \leftarrow \mathcal{D}_t$
**end for**
**return** best observed incumbent $\boldsymbol{x}^\star$

predicted Hessian is therefore symmetrized and modified by shifting or clipping its eigenvalues so that the resulting matrix $\tilde{H}_t$ is negative definite. The search direction is then

$$\boldsymbol{p}_t = -\tilde{\boldsymbol{H}}_t^{-1}\boldsymbol{g}_t.$$

To avoid always taking the full Newton-type step, PrefSQP performs a one-dimensional preference-based line search over

$$\boldsymbol{x}_t(\eta) = \Pi_\Theta(\boldsymbol{x}_t + \eta \boldsymbol{p}_t), \qquad \eta \in [0, 1].$$

In our implementation, $[0, 1]$ is discretized into a grid of candidate step sizes and Thompson sampling is used to select the next point along this line. The selected point is then used as the next iterate according to the preference-based line-search rule. The full procedure is summarized in Algorithm 2.

### B.5 TuRPBO: Trust-Region Preferential Bayesian Optimization

TuRPBO follows the trust-region adaptation rules of TuRBO, but replaces scalar improvement by pairwise preference improvement. Let $\boldsymbol{x}_t^\star$ be the current incumbent and $B_t = \{\boldsymbol{x}_{t,1}, \dots, \boldsymbol{x}_{t,q}\}$ the batch selected inside the trust region. An iteration is successful if at least one candidate is preferred to the incumbent,

$$\exists \boldsymbol{x}_{t,i} \in B_t \quad \text{s.t.} \quad \boldsymbol{x}_{t,i} \succ \boldsymbol{x}_t^\star.$$

In that case, the incumbent is updated to the preferred candidate, the success counter is incremented, and the failure counter is reset. Otherwise, the incumbent remains unchanged, the failure counter is incremented, and the success counter is reset. After $\tau_{\mathrm{succ}}$ consecutive successes, the trust-region length is doubled up to $L_{\max}$; after $\tau_{\mathrm{fail}}$ consecutive failures, it is halved. If the length drops below $L_{\min}$, the trust region is restarted.

### B.5.1 FULL ALGORITHM

**Algorithm 3** TurPBO

Initialize data set $\mathcal{D}_0$, incumbent $\boldsymbol{x}_0^\star$, trust-region length $L_0 = L_{\text{init}}$, $\tau_{\text{fail}}$ and $\tau_{\text{succ}}$
Set $c_{\text{succ}} = 0$, $c_{\text{fail}} = 0$
**for** $t = 0, \dots, T-1$ **do**
  Fit pairwise GP model using $\mathcal{D}_t$
  Construct trust region $\mathcal{T}_t$ around $\boldsymbol{x}_t^\star$
  Select batch $B_t = \{\boldsymbol{x}_{t,1}, \dots, \boldsymbol{x}_{t,q}\} \subset \mathcal{T}_t$ using TS or qEUBO
  Compare candidates in $B_t$ against incumbent $\boldsymbol{x}_t^\star$
  **if** some $\boldsymbol{x}_{t,i} \in B_t$ is preferred to $\boldsymbol{x}_t^\star$ **then**
    Set $\boldsymbol{x}_{t+1}^\star$ to the preferred candidate
    $c_{\text{succ}} \leftarrow c_{\text{succ}} + 1$, $c_{\text{fail}} \leftarrow 0$
  **else**
    $\boldsymbol{x}_{t+1}^\star \leftarrow \boldsymbol{x}_t^\star$
    $c_{\text{fail}} \leftarrow c_{\text{fail}} + 1$, $c_{\text{succ}} \leftarrow 0$
  **end if**
  **if** $c_{\text{succ}} = \tau_{\text{succ}}$ **then**
    $L_{t+1} \leftarrow \min(2L_t, L_{\max})$
    Reset $c_{\text{succ}}$ and $c_{\text{fail}}$
  **else if** $c_{\text{fail}} = \tau_{\text{fail}}$ **then**
    $L_{t+1} \leftarrow L_t/2$
    Reset $c_{\text{succ}}$ and $c_{\text{fail}}$
  **else**
    $L_{t+1} \leftarrow L_t$
  **end if**
  **if** $L_{t+1} < L_{\min}$ **then**
    Restart trust region around a new incumbent
  **end if**
  Update $\mathcal{D}_{t+1}$ with the newly observed comparisons
**end for**

# C Experimental Details and Additional Results

## C.1 Shared Experimental Setup

**Evaluation and comparison budgets.** We report budgets in terms of function evaluations, since these are the expensive operations in our target applications. In the pairwise setting, the number of comparisons is nevertheless almost identical to the number of function evaluations: after the two initial evaluations, every newly evaluated point is compared against an incumbent. Since function values are not observed by the algorithms, the incumbent is defined purely from preference data as the current winner under the observed comparisons. For most methods this is the best-seen point so far, updated whenever a newly evaluated point is preferred to the current incumbent. Thus, $N$ function evaluations yield $N-1$ pairwise comparisons. For the gradient-based methods, comparisons are made only during the local gradient-estimation phases against a local incumbent; the subsequent center updates are not immediately evaluated. Hence, if $G$ gradient steps are taken, $N$ function evaluations yield $N-G$ pairwise comparisons. Thus, a budget such as $2+10d$ refers to function evaluations, not acquisition steps, but corresponds to approximately the same number of pairwise comparisons. In addition to the baselines in the main paper, we also use GLISp (Bemporad, 2020) as a non-BO baseline in the appendix.

Table 1: Experiment-level settings used across benchmark experiments. Values marked "experiment-specific" are reported in the corresponding benchmark sections.

| Setting | Value / Description |
|---|---|
| Number of independent runs | 10 |
| Initial design size for synthetic and GP benchmarks | 2 random points |
| Initial design size for policy-search benchmarks | $5d$ random evaluations |
| Optimization budget for GP benchmarks | $2+10d$ evaluations |
| Optimization budget for synthetic benchmarks | $2+10d$ evaluations |
| Optimization budget for policy-search benchmarks | $5d+10d$ evaluations |
| Input normalization | All benchmark domains mapped to $[0,1]^d$ |
| Observation type given to algorithms | Noisy pairwise comparisons only |
| Noise model | Gaussian noise with standard deviation equal to $10\%$ of the approximate function-value range |
| Reported metric | Best-seen value $f_{\text{best}}$ and cumulative performance $\sum_k f_k$ |
| Uncertainty visualization | Median with 25/75 percentiles, unless stated otherwise |
| Hyperparameter learning | Experiment-specific; known for within-model GP experiments and learned for out-of-model/synthetic/policy experiments |
| Baselines | Sobol, GLISp, qEUBO, LineCoSpar |
| Local methods | (G2L)-GIPBO, (G2L)-GIPBOr, PrefSQP, TuRPBO |

## C.2 Shared PairwiseGP Settings

Note that the `PairwiseGP` implementation in BoTorch, used for local PBO methods and EUBO, implicitly assumes that the observation noise $\sigma$ equals 1. Different noise levels are therefore represented through the kernel output scale. We follow this convention when computing preference likelihoods for the conditional expectation in the adapted GI acquisition function.

Table 2: Shared surrogate-model and acquisition-optimization settings for PairwiseGP-based methods. These settings apply to EUBO, GIPBO, GIPBOr, PrefSQP, and TuRPBO unless stated otherwise.

| **Hyperparameter** | **Value / Description** |
|---|---|
| GP model | BoTorch `PairwiseGP` |
| Kernel | ScaleKernel with squared-exponential base kernel |
| Input domain | Unit hypercube $[0, 1]^d$ |
| Lengthscale constraint | $[0.05, 0.5]$ |
| Preference noise convention | BoTorch convention with implicit $\sigma = 1$ and adjusted output scale |
| Acquisition optimization restarts | 5 |
| Raw samples for acquisition optimization | 1024 |
| Batch size $q$ | 1 unless stated otherwise |

### C.3 Algorithm Configurations

Table 3: Hyperparameters and implementation choices for GLISp.

| **Hyperparameter** | **Value / Description** |
|---|---|
| Comparison tolerance | $10^{-4}$ |
| Initial random points for virtual initialization | 10 |
| Lower bound | 0 in each dimension |
| Upper bound | 1 in each dimension |
| Input domain | Unit hypercube $[0, 1]^d$ |

Table 4: Hyperparameters and implementation choices for GIPBO and GIPBOr.

| **Hyperparameter** | **Value / Description** |
|---|---|
| Gradient step length $\eta$ | 0.5 |
| Minimum absolute step length | 0.05 |
| Number of local queries per gradient estimate $M$ | $d$ |
| Number of gradient steps $N$ | 10 |
| Acquisition search bounds $(\delta_{\min}, \delta_{\max})$ | $[0.05, 0.2]$ |
| Resolution for $d > 2$ | 50 |
| Hypersphere radii for $d > 2$ | $[0.05, 0.1, 0.15, 0.2]$ |
| Virtual preference likelihood threshold | 0.1 |
| Gradient normalization | Mahalanobis norm induced by SE lengthscales |
| GIPBOr acquisition selection | Random samples within the same local search bounds |

Table 5: Hyperparameters and implementation choices for PrefSQP.

| **Hyperparameter** | **Value / Description** |
|---|---|
| Local sampling radius $\epsilon$ | 0.05 |
| Number of local samples per SQP step | $d$ |
| Hessian modification | Eigenvalue modification to enforce negative definiteness |
| Line-search domain | $\eta \in [0, 1]$ |
| Line-search resolution | 100 |
| Line-search acquisition | Thompson sampling unless stated otherwise |
| Step direction | Newton-type direction from posterior mean gradient and Hessian |

Table 6: Hyperparameters and implementation choices for TuRPBO. Here, $d$ denotes the problem dimensionality and $q$ the batch size.

| **Hyperparameter** | **Value / Description** |
|---|---|
| Batch size $q$ | 1 |
| Acquisition function | Thompson sampling unless stated otherwise |
| Acquisition optimization restarts | 5 |
| Raw samples for acquisition optimization | 1024 |
| Initial trust-region length $L_{\text{init}}$ | 0.8 |
| Minimum trust-region length $L_{\min}$ | $0.5^7$ |
| Maximum trust-region length $L_{\max}$ | 1.6 |
| Success tolerance $\tau_{\text{succ}}$ | $d$ |
| Failure tolerance $\tau_{\text{fail}}$ | $\lceil \max(4/q, d/q) \rceil$ |
| Trust-region expansion | $L \leftarrow \min(2L, L_{\max})$ |
| Trust-region contraction | $L \leftarrow L/2$ |
| Restart criterion | Restart if $L < L_{\min}$ |

### C.4 GP Sample-Path Experiments

In addition to the plots in Figure 2, that only shows the final performances for the different algorithms in the within and out-of-model comparison, Figure 7 and 8 show the full performance curves for these experiments. In the experiments, we sampled a new GP sample for each run with a random initialization. All algorithms used the same initial points per run.

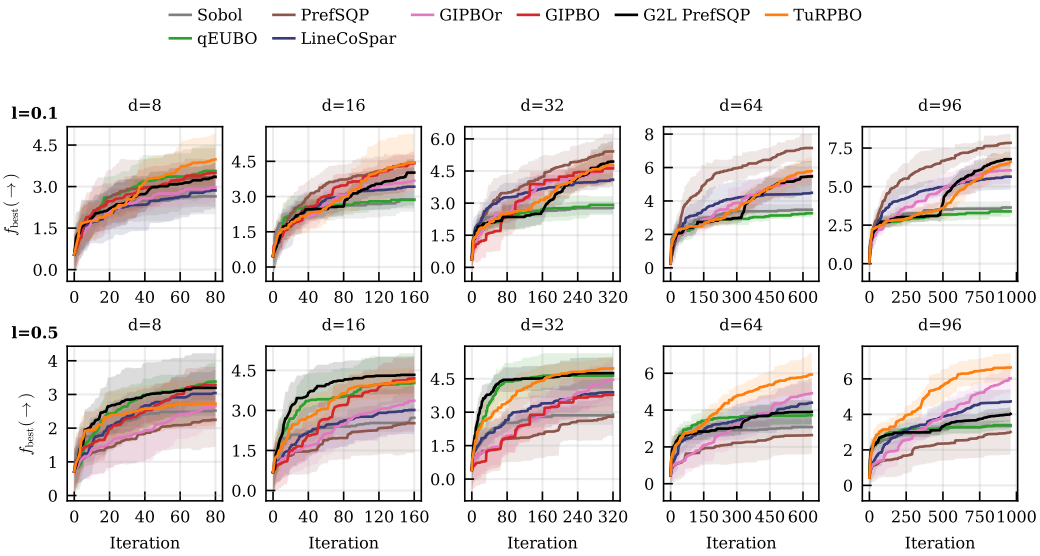



Figure 7: Within-model comparison. Performance over iterations for GP samples between 8 and 96$d$.

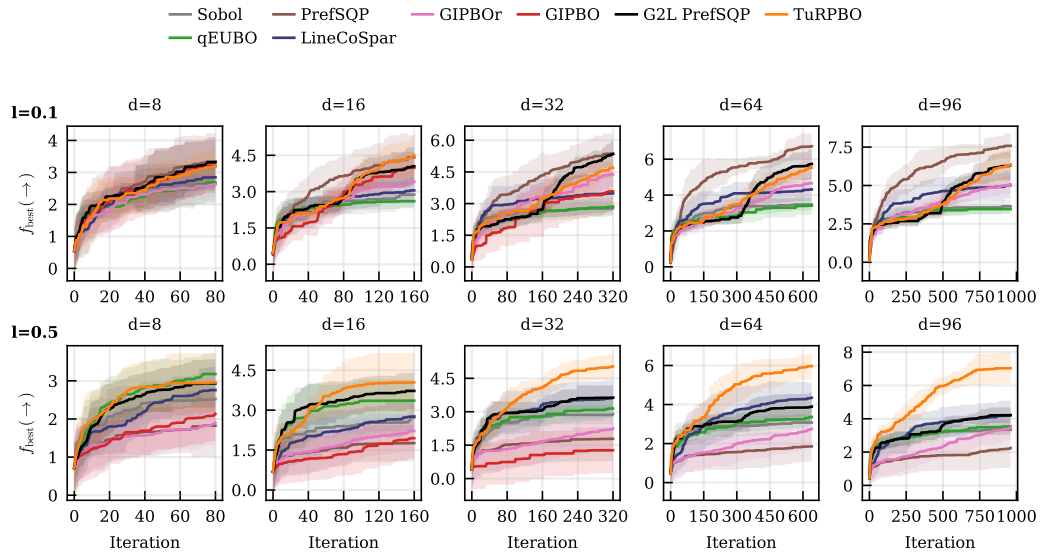



Figure 8: Out-of-model comparison. Performance over iterations for GP samples between 8 and 96$d$.

GIPBOr performs worse than full GIPBO in 8 and 16 dimensions, especially for short lengthscales. In 32 dimensions, this advantage largely vanishes. This is caused by the implementation cap on the number of evaluated acquisition candidates: evaluating the full preferential GI acquisition becomes expensive, and the capped candidate set is too sparse to reliably approximate the acquisition maximizer in higher dimensions.

Comparing the within-model and out-of-model settings, GIPBO and GIPBOr in particular degrade in performance when the model is misspecified. This can be explained by the fact that the gradient step is scaled using the lengthscale as described in B.2.

Furthermore, LineCoSpar is a competitive baseline, achieving comparable performance in high-dimensional settings with long length scales and outperforming G2L-PrefSQP in the within-model comparison.

### C.5 Standard Benchmark Details

For the synthetic benchmark experiments, we use the test functions from the BoTorch library Balandat et al. (2020). All benchmark domains are normalized to $[0, 1]^d$ and all algorithms are initialized with the same two random points in each run. The optimization budget is $2 + 10d$ evaluations. The algorithms observe only noisy pairwise comparisons, where the noise standard deviation is set to $10\%$ of the approximate function-value range. The main synthetic benchmark results are shown in Figure 3 and the compute times are reported in Table 12.

For readability, the main plots report only mean $\pm$ std; here we provide detailed numerical results and statistical significance tests. We use a pairwise Wilcoxon signed-rank test (Wilcoxon, 1945) to compare our results similar to the test in Erarslan et al. (2025). As we perform multiple tests, we apply a Holm correction to the results. We always test if the best performing algorithm (bold in the plot) performs significantly different from the others. Algorithms which are not significantly performing differently from the best performing one are underlined in the plot.

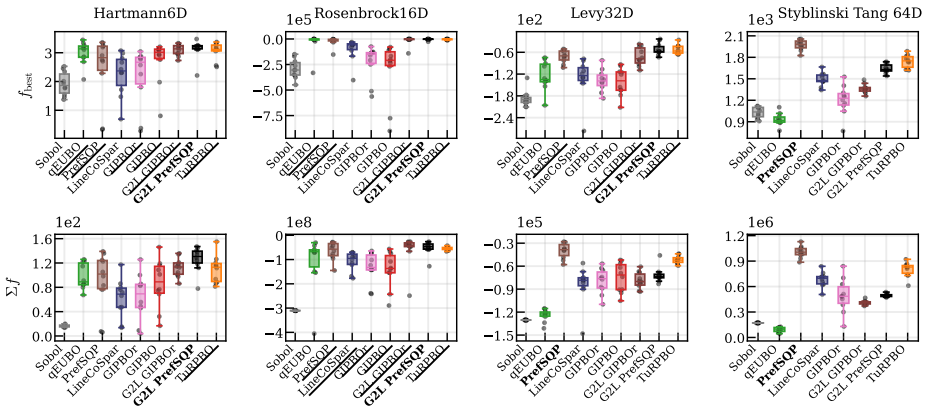



Figure 9: Final performance boxplots for the different optimization functions. The best performing algorithm has a bold label. The ones that are significantly different have a normal label, the ones that are not statistically different are underlined.

### C.6 Policy-Search

For the policy search task we use the implementation from ARS Mania et al. (2018). We warmstart the experiment by selecting the best point out of $5d$ iterations and comparing it to a random value to emulate the case of a promising starting point. In Table 7, we report the results and use a paired Wilcoxon as in the previous section, to determine whether the best performing algorithm significantly outperforms the baselines. The bold value in the table is the best algorithm, the underlined values are not significantly different.

Table 7: Final and cumulative benchmark performance in the policy search task with warmstarting, the bold label is the best for each metric, the underlined values do not differ significantly in the statistical test.

| Metric | Function | Sobol | qEUBO | PrefSQP | LineCoSpar | GIPBOr | G2L GIPBOr | G2L PrefSQP | TuRPBO |
|---|---|---|---|---|---|---|---|---|---|
| $f_{\text{best}} \uparrow$ | HopperLinearPolicyTask 33D | 313 ± 61 | 317 ± 44 | 306 ± 90 | 345 ± 54 | 334 ± 82 | 347 ± 53 | 347 ± 77 | 413 ± 58 |
| $f_{\text{best}} \uparrow$ | Walker2dLinearPolicyTask 102D | 102 ± 27 | 124 ± 49 | 201 ± 56 | 135 ± 81 | 247 ± 95 | 189 ± 84 | 224 ± 120 | **330 ± 134** |
| $\sum_k f_k \uparrow$ | HopperLinearPolicyTask 33D | 4962 ± 396 | 6865 ± 2440 | 55870 ± 20064 | 47872 ± 16391 | **57747 ± 22071** | 40978 ± 12631 | 39637 ± 12144 | 54489 ± 10398 |
| $\sum_k f_k \uparrow$ | Walker2dLinearPolicyTask 102D | -6589 ± 175 | -6868 ± 325 | 19283 ± 13740 | 30898 ± 24715 | 33207 ± 25362 | 16560 ± 9808 | 17049 ± 12358 | **62616 ± 51688** |

### C.7 Ablations

For the different algorithm variants, we perform an ablation in $32d$. We conduct the experiments in an out-of-model comparison on $32d$ GP samples with lengthscales $0.1$ and $0.5$, on the Levy function in $32d$ as well as the Styblinsky-Tang $32d$ test function.

For PrefSQP, we ablate the different variants with different acquisition options for the gradient: using the suggested step length (fixed, $\eta = 1$), using line search with EUBO or with Thompson sampling. Furthermore, we investigate if a Newton step or the uncertainty aware subproblem from Brunzema & Trimpe (2025) should be used (EV). Figure 10 shows that in the synthetic setting, all variants perform similarly. On the GP functions, the fixed Newton step works best.

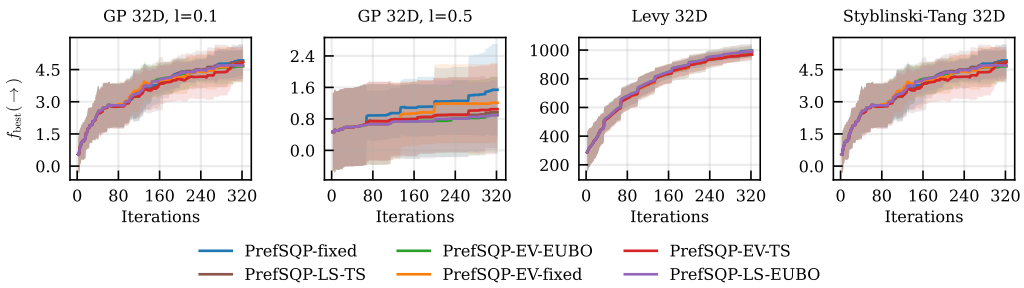



Figure 10: PrefSQP variant ablation

For TuRPBO, we also compare different acquisition strategies. We compare the perturbed Thompson sampling variant, which is our base TuRPBO version, to expected improvement and qEUBO Astudillo et al. (2023). On the GP samples TuRPBO performs best, while on the standard benchmarking functions, the qEUBO variant performs equivalently well.

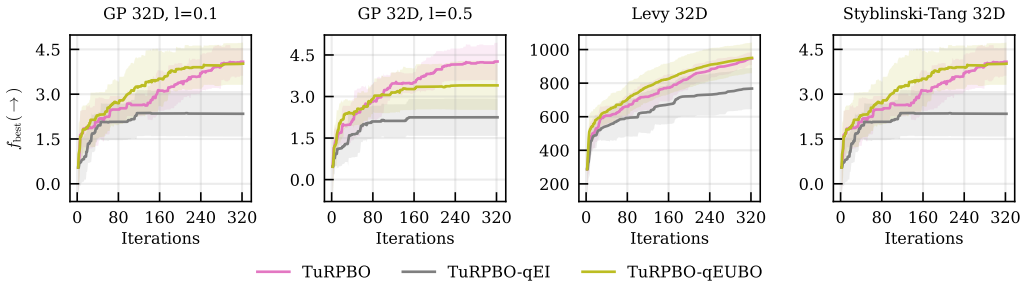



Figure 11: Different Acquisition strategies in TuRPBO

In all our settings, we compare against qEUBO with batch size one. However, a key selling point of qEUBO is, that the batch size can be increased and therefore the number of comparisons can be reduced. In this paper, we focus on the number of experiments, as we assume that in a policy searching task the environment interactions are more expensive than the comparisons. Nevertheless, we also compare the performance of different qEUBO batch sizes as a reference in Figure 12.

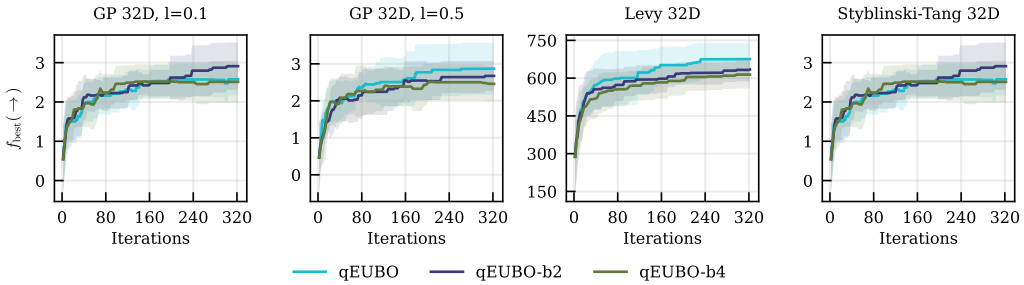



Figure 12: Comparison of qEUBO batch size 1, 2 and 4

In our benchmarking setup, all qEUBO variants perform equally over the number of experiments, however batch size two would only need half the number of decisions and batch size four only a fourth. However, the final performance after the number of iterations in the $32d$ case is still lower than all in the local variants we propose.

### C.8 Real-World Use Cases

In addition to the synthetic functions, we also evaluate our methods on the Sushi (4D) (Siivola et al., 2021) and CarCab (7D) (Lin et al., 2022) tasks, which are common benchmarks for PBO. In contrast to our other experiments, these tasks are low-dimensional, but the noise arises from human preferences rather than injected Gaussian noise. We also ablate the initialization strategy by comparing two schemes. In the random-comparison setting, we use the default baseline with $n = 2$, as well as variants with $d$, $2 \times d$, and $4 \times d$ initial random pairwise comparisons, corresponding to $d$, $2 \times d$, and $4 \times d$ initial points. In the anchor setting, we first evaluate $d$, $2d$, or $4d$ randomly sampled candidate points and select the best as an anchor. This anchor is then paired with one additional random point, so the stored initial dataset still contains two points while benefiting from the preliminary random search. After initialization, we perform $10 \times \mathrm{dim}$ experiments. Compared with Astudillo et al. (2023), who initialize with $4 \times \mathrm{dim}$ points and then use 150 queries corresponding to 2 or 4 function evaluations each, our setup uses fewer function evaluations.

Table 8: Performance on the CarCab 7D benchmark.

| Algorithm | $n_{\text{init}} = 2$ | Comparisons | | | Anchor | | |
|---|---|---|---|---|---|---|---|
| | Baseline | $1 \times d$ | $2 \times d$ | $4 \times d$ | $1 \times d$ | $2 \times d$ | $4 \times d$ |
| Sobol | $-2.503 \pm 0.171$ | $-2.482 \pm 0.153$ | $-2.475 \pm 0.140$ | $-2.490 \pm 0.121$ | $-2.484 \pm 0.156$ | $-2.516 \pm 0.187$ | $-2.457 \pm 0.139$ |
| GLISp | $-2.733 \pm 0.669$ | $-2.217 \pm 0.248$ | $-2.191 \pm 0.195$ | $-2.205 \pm 0.182$ | $-2.405 \pm 0.300$ | $-2.321 \pm 0.160$ | $-2.305 \pm 0.158$ |
| qEUBO | $-2.460 \pm 0.545$ | $-2.326 \pm 0.320$ | $-2.376 \pm 0.301$ | $-2.174 \pm 0.206$ | $-2.269 \pm 0.347$ | $-2.146 \pm 0.260$ | $-2.184 \pm 0.272$ |
| PrefSQP | $-2.984 \pm 0.491$ | $-2.408 \pm 0.197$ | $-2.368 \pm 0.125$ | $-2.326 \pm 0.134$ | $-2.552 \pm 0.270$ | $-2.462 \pm 0.238$ | $-2.415 \pm 0.131$ |
| LineCoSpar | $-2.402 \pm 0.341$ | $-2.341 \pm 0.166$ | $-2.333 \pm 0.172$ | $-2.317 \pm 0.108$ | $-2.321 \pm 0.168$ | $-2.372 \pm 0.199$ | $-2.335 \pm 0.134$ |
| GIPBOr | $-3.081 \pm 0.479$ | $-2.532 \pm 0.137$ | $-2.523 \pm 0.235$ | $-2.293 \pm 0.163$ | $-2.446 \pm 0.283$ | $-2.368 \pm 0.180$ | $-2.276 \pm 0.143$ |
| GIPBO | $-3.049 \pm 0.460$ | $-2.551 \pm 0.191$ | $-2.545 \pm 0.243$ | $-2.307 \pm 0.187$ | $-2.488 \pm 0.285$ | $-2.353 \pm 0.219$ | $-2.271 \pm 0.206$ |
| G2L GIPBOr | $-2.476 \pm 0.536$ | $-2.318 \pm 0.247$ | $-2.277 \pm 0.214$ | $-2.131 \pm 0.144$ | $-2.257 \pm 0.319$ | $-2.123 \pm 0.218$ | $-2.170 \pm 0.234$ |
| G2L GIPBO | $-2.548 \pm 0.577$ | $-2.355 \pm 0.338$ | $-2.264 \pm 0.231$ | $-2.147 \pm 0.179$ | $-2.276 \pm 0.341$ | $-2.100 \pm 0.200$ | $-2.175 \pm 0.227$ |
| G2L PrefSQP | $\mathbf{-2.189 \pm 0.313}$ | $-2.355 \pm 0.191$ | $-2.338 \pm 0.210$ | $-2.160 \pm 0.190$ | $-2.184 \pm 0.188$ | $-2.161 \pm 0.256$ | $-2.177 \pm 0.195$ |
| TuRPBO | $-2.223 \pm 0.141$ | $-2.280 \pm 0.158$ | $-2.239 \pm 0.177$ | $-2.146 \pm 0.103$ | $-2.231 \pm 0.164$ | $-2.213 \pm 0.242$ | $-2.277 \pm 0.173$ |

Table 9: Cumulative performance on the CarCab 7D benchmark.

| Algorithm | $n_{\text{init}} = 2$ | Comparisons | | | Anchor | | |
|---|---|---|---|---|---|---|---|
| | Baseline | $1 \times d$ | $2 \times d$ | $4 \times d$ | $1 \times d$ | $2 \times d$ | $4 \times d$ |
| Sobol | $-385.311 \pm 1.011$ | $-445.451 \pm 7.106$ | $-521.498 \pm 11.620$ | $-675.524 \pm 11.052$ | $-382.073 \pm 1.080$ | $-382.653 \pm 2.443$ | $-383.200 \pm 2.433$ |
| GLISp | $-285.722 \pm 67.235$ | $-280.877 \pm 64.787$ | $-325.503 \pm 38.530$ | $-470.729 \pm 26.529$ | $-300.518 \pm 79.685$ | $-286.458 \pm 82.511$ | $-275.728 \pm 73.854$ |
| qEUBO | $-249.229 \pm 43.516$ | $-319.815 \pm 41.653$ | $-403.957 \pm 57.131$ | $-536.795 \pm 23.609$ | $-239.664 \pm 64.203$ | $-248.707 \pm 43.443$ | $-262.007 \pm 63.296$ |
| PrefSQP | $-251.315 \pm 38.063$ | $-254.649 \pm 22.738$ | $-321.934 \pm 18.431$ | $-474.281 \pm 16.828$ | $-196.959 \pm 19.989$ | $-190.974 \pm 18.203$ | $-186.622 \pm 9.300$ |
| LineCoSpar | $-245.126 \pm 19.465$ | $-290.374 \pm 22.159$ | $-379.410 \pm 21.355$ | $-536.340 \pm 22.803$ | $-214.022 \pm 19.489$ | $-218.464 \pm 23.148$ | $-212.681 \pm 15.568$ |
| GIPBOr | $-268.543 \pm 46.856$ | $-299.136 \pm 41.783$ | $-370.481 \pm 58.453$ | $-485.857 \pm 29.863$ | $-195.103 \pm 19.527$ | $-191.584 \pm 18.053$ | $-182.460 \pm 9.848$ |
| GIPBO | $-263.505 \pm 44.597$ | $-299.461 \pm 46.784$ | $-366.688 \pm 54.862$ | $-484.736 \pm 27.676$ | $-194.236 \pm 21.825$ | $-189.409 \pm 16.892$ | $-179.305 \pm 11.973$ |
| G2L GIPBOr | $-223.521 \pm 40.553$ | $-276.649 \pm 26.362$ | $-352.541 \pm 41.938$ | $-480.786 \pm 22.243$ | $-199.794 \pm 31.517$ | $-195.252 \pm 20.187$ | $-201.691 \pm 29.500$ |
| G2L GIPBO | $-222.968 \pm 40.803$ | $-275.697 \pm 27.525$ | $-351.109 \pm 41.072$ | $-479.464 \pm 22.210$ | $-197.997 \pm 32.069$ | $-193.551 \pm 19.196$ | $-199.929 \pm 29.201$ |
| G2L PrefSQP | $-205.336 \pm 23.119$ | $-275.049 \pm 29.847$ | $-349.610 \pm 36.425$ | $-477.340 \pm 22.584$ | $-189.942 \pm 15.214$ | $-194.381 \pm 27.162$ | $-203.572 \pm 42.894$ |
| TuRPBO | $\mathbf{-196.136 \pm 14.489}$ | $-249.839 \pm 15.008$ | $-320.928 \pm 12.609$ | $-467.021 \pm 12.238$ | $-187.432 \pm 11.218$ | $-183.356 \pm 16.154$ | $-185.061 \pm 10.241$ |

Table 10: Performance on the Sushi 4D benchmark.

| Algorithm | $n_{\text{init}} = 2$ | Comparisons | | | Anchor | | |
|---|---|---|---|---|---|---|---|
| | Baseline | $1 \times d$ | $2 \times d$ | $4 \times d$ | $1 \times d$ | $2 \times d$ | $4 \times d$ |
| Sobol | $\mathbf{-0.156 \pm 0.062}$ | $-0.171 \pm 0.070$ | $-0.186 \pm 0.075$ | $-0.167 \pm 0.060$ | $\mathbf{-0.134 \pm 0.041}$ | $-0.158 \pm 0.083$ | $-0.147 \pm 0.076$ |
| GLISp | $-0.410 \pm 0.212$ | $-0.181 \pm 0.084$ | $-0.166 \pm 0.084$ | $-0.160 \pm 0.094$ | $-0.260 \pm 0.149$ | $-0.161 \pm 0.101$ | $-0.158 \pm 0.087$ |
| qEUBO | $-0.206 \pm 0.080$ | $-0.175 \pm 0.078$ | $-0.146 \pm 0.071$ | $-0.137 \pm 0.059$ | $-0.190 \pm 0.067$ | $-0.164 \pm 0.134$ | $-0.127 \pm 0.072$ |
| PrefSQP | $-0.311 \pm 0.141$ | $-0.150 \pm 0.056$ | $-0.145 \pm 0.070$ | $-0.135 \pm 0.069$ | $-0.290 \pm 0.119$ | $-0.156 \pm 0.089$ | $-0.150 \pm 0.078$ |
| LineCoSpar | $-0.260 \pm 0.158$ | $\mathbf{-0.119 \pm 0.055}$ | $-0.144 \pm 0.069$ | $-0.169 \pm 0.058$ | $-0.195 \pm 0.096$ | $-0.144 \pm 0.042$ | $-0.185 \pm 0.101$ |
| GIPBOr | $-0.238 \pm 0.137$ | $-0.174 \pm 0.070$ | $-0.145 \pm 0.065$ | $-0.151 \pm 0.086$ | $-0.246 \pm 0.130$ | $-0.132 \pm 0.072$ | $-0.142 \pm 0.082$ |
| GIPBO | $-0.309 \pm 0.140$ | $-0.166 \pm 0.074$ | $\mathbf{-0.120 \pm 0.064}$ | $-0.135 \pm 0.049$ | $-0.226 \pm 0.076$ | $-0.150 \pm 0.139$ | $-0.136 \pm 0.096$ |
| G2L GIPBOr | $-0.186 \pm 0.072$ | $-0.149 \pm 0.077$ | $-0.149 \pm 0.070$ | $-0.142 \pm 0.082$ | $-0.215 \pm 0.092$ | $-0.141 \pm 0.118$ | $\mathbf{-0.117 \pm 0.070}$ |
| G2L GIPBO | $-0.197 \pm 0.092$ | $-0.168 \pm 0.073$ | $-0.139 \pm 0.068$ | $-0.142 \pm 0.086$ | $-0.234 \pm 0.148$ | $-0.155 \pm 0.136$ | $-0.123 \pm 0.062$ |
| G2L PrefSQP | $-0.256 \pm 0.122$ | $-0.144 \pm 0.060$ | $-0.139 \pm 0.067$ | $\mathbf{-0.128 \pm 0.064}$ | $-0.283 \pm 0.146$ | $-0.184 \pm 0.139$ | $-0.154 \pm 0.123$ |
| TuRPBO | $-0.173 \pm 0.102$ | $-0.160 \pm 0.065$ | $-0.135 \pm 0.060$ | $-0.133 \pm 0.063$ | $-0.202 \pm 0.152$ | $\mathbf{-0.114 \pm 0.059}$ | $-0.161 \pm 0.083$ |

Table 11: Cumulative performance on the Sushi 4D benchmark.

| Algorithm | $n_{\text{init}} = 2$ | Comparisons | | | Anchor | | |
|---|---|---|---|---|---|---|---|
| | Baseline | $1 \times d$ | $2 \times d$ | $4 \times d$ | $1 \times d$ | $2 \times d$ | $4 \times d$ |
| Sobol | $-26.460 \pm 1.135$ | $-30.141 \pm 1.375$ | $-35.078 \pm 1.522$ | $-44.777 \pm 2.559$ | $-26.178 \pm 1.316$ | $-26.181 \pm 1.273$ | $-26.149 \pm 1.054$ |
| GLISp | $-23.130 \pm 9.566$ | $\mathbf{-14.856 \pm 4.233}$ | $-20.388 \pm 5.693$ | $-29.822 \pm 6.508$ | $-18.322 \pm 6.545$ | $-15.101 \pm 7.106$ | $-13.632 \pm 3.969$ |
| qEUBO | $-19.433 \pm 4.258$ | $-22.051 \pm 4.198$ | $-25.674 \pm 3.726$ | $-34.503 \pm 4.288$ | $-21.836 \pm 4.669$ | $-20.794 \pm 5.110$ | $-18.614 \pm 4.442$ |
| PrefSQP | $-18.174 \pm 5.918$ | $-14.984 \pm 3.700$ | $\mathbf{-18.829 \pm 3.885}$ | $\mathbf{-28.802 \pm 4.044}$ | $-15.912 \pm 5.862$ | $\mathbf{-9.783 \pm 3.421}$ | $-9.633 \pm 3.953$ |
| LineCoSpar | $-22.563 \pm 4.842$ | $-20.829 \pm 2.726$ | $-26.532 \pm 3.551$ | $-36.287 \pm 3.143$ | $-20.080 \pm 5.179$ | $-16.768 \pm 3.253$ | $-15.948 \pm 4.678$ |
| GIPBOr | $-17.293 \pm 5.700$ | $-22.133 \pm 6.825$ | $-22.754 \pm 8.853$ | $-31.685 \pm 6.092$ | $-17.397 \pm 7.870$ | $-10.938 \pm 4.192$ | $-10.096 \pm 4.145$ |
| GIPBO | $-19.418 \pm 5.564$ | $-21.099 \pm 6.491$ | $-23.002 \pm 9.389$ | $-31.107 \pm 3.138$ | $-17.194 \pm 5.665$ | $-10.614 \pm 4.599$ | $\mathbf{-9.475 \pm 3.829}$ |
| G2L GIPBOr | $-18.108 \pm 3.070$ | $-18.543 \pm 3.475$ | $-21.648 \pm 2.558$ | $-31.793 \pm 4.907$ | $-17.707 \pm 4.765$ | $-15.752 \pm 4.123$ | $-14.502 \pm 4.479$ |
| G2L GIPBO | $-17.936 \pm 2.960$ | $-18.180 \pm 3.578$ | $-21.298 \pm 3.008$ | $-31.479 \pm 4.731$ | $-17.149 \pm 5.207$ | $-15.293 \pm 4.540$ | $-13.742 \pm 4.637$ |
| G2L PrefSQP | $-18.036 \pm 4.752$ | $-17.075 \pm 2.246$ | $-21.521 \pm 3.324$ | $-30.295 \pm 3.241$ | $-19.138 \pm 4.731$ | $-15.270 \pm 4.836$ | $-13.667 \pm 5.561$ |
| TuRPBO | $\mathbf{-15.214 \pm 3.313}$ | $-16.079 \pm 3.283$ | $-19.943 \pm 3.432$ | $-29.820 \pm 4.570$ | $\mathbf{-14.608 \pm 4.678}$ | $-12.024 \pm 2.056$ | $-12.462 \pm 3.534$ |

## D Computational Resources

All experiments were conducted using CPUs only on a high-performance computing (HPC) cluster. At the time of the experiments, the cluster ran *Rocky Linux 9.6*. Jobs were submitted via Slurm to the `c23mm` partition. Each run was executed on a single node with one Slurm task and eight CPU cores allocated to that task, with `5000M` of memory requested per CPU core.

Table 12: Compute time per algorithm across benchmark functions.

| | Global methods | | | Local methods | | | | |
|---|---|---|---|---|---|---|---|---|
| Function | Sobol | qEUBO ($q = 1$) | LineCoSpar | GIPBO | GIPBOr | TuRPBO | PrefSQP | G2L-PrefSQP |
| Hartmann 6D | $0.022 \pm 0.001$ | $31 \pm 5$ | $10 \pm 2$ | $245 \pm 23$ | $2.6 \pm 0.8$ | $44 \pm 4$ | $20 \pm 17$ | $23 \pm 3$ |
| Rosenbrock 16D | $0.058 \pm 0.007$ | $243 \pm 195$ | $39 \pm 8$ | $2453 \pm 305$ | $9.3 \pm 10.2$ | $182 \pm 108$ | $92 \pm 68$ | $192 \pm 145$ |
| Levy 32D | $0.15 \pm 0.01$ | $783 \pm 517$ | $119 \pm 35$ | $15556 \pm 2408$ | $14 \pm 15$ | $557 \pm 156$ | $457 \pm 102$ | $613 \pm 674$ |
| StyblinskiTang 64D | $0.73 \pm 0.62$ | $4975 \pm 1912$ | $802 \pm 400$ | – | $36 \pm 8$ | $2971 \pm 606$ | $2926 \pm 580$ | $4832 \pm 940$ |

The compute time for the main results are reported as average per run in Table 12, Table 13 and Table 14.

Table 13: Compute time per algorithm across benchmark functions.

| | Global methods | | | Local methods | | | | |
|---|---|---|---|---|---|---|---|---|
| Function | Sobol | qEUBO ($q = 1$) | LineCoSpar | GIPBO | GIPBOr | TuRPBO | PrefSQP | G2L-PrefSQP |
| GPSample 8D ls=0.1 (within) | $0.039 \pm 0.003$ | $14 \pm 10$ | $4.6 \pm 1.5$ | $311 \pm 24$ | $1.3 \pm 1.0$ | $45 \pm 3$ | $6.7 \pm 1.8$ | $14 \pm 7$ |
| GPSample 8D ls=0.5 (within) | $0.037 \pm 0.001$ | $19 \pm 5$ | $9.7 \pm 8.9$ | $532 \pm 232$ | $1.1 \pm 0.4$ | $47 \pm 3$ | $12 \pm 9$ | $19 \pm 12$ |
| GPSample 16D ls=0.1 (within) | $0.099 \pm 0.022$ | $75 \pm 26$ | $9.7 \pm 2.2$ | $1794 \pm 105$ | $1.6 \pm 0.5$ | $101 \pm 14$ | $21 \pm 4$ | $79 \pm 66$ |
| GPSample 16D ls=0.5 (within) | $0.097 \pm 0.022$ | $96 \pm 30$ | $20 \pm 4$ | $2900 \pm 472$ | $2.1 \pm 0.6$ | $103 \pm 10$ | $30 \pm 15$ | $61 \pm 21$ |
| GPSample 32D ls=0.1 (within) | $0.42 \pm 0.65$ | $341 \pm 132$ | $38 \pm 10$ | $28955 \pm 15083$ | $3.5 \pm 0.8$ | $345 \pm 125$ | $117 \pm 50$ | $238 \pm 94$ |
| GPSample 32D ls=0.5 (within) | $0.39 \pm 0.63$ | $193 \pm 64$ | $49 \pm 8$ | $34879 \pm 6996$ | $3.9 \pm 0.7$ | $252 \pm 31$ | $147 \pm 51$ | $171 \pm 61$ |
| GPSample 64D ls=0.1 (within) | $0.49 \pm 0.16$ | $13171 \pm 3467$ | $156 \pm 39$ | – | $12 \pm 4$ | $1564 \pm 604$ | $839 \pm 172$ | $3886 \pm 1144$ |
| GPSample 64D ls=0.5 (within) | $0.49 \pm 0.07$ | $1707 \pm 1100$ | $279 \pm 106$ | – | $17 \pm 3$ | $1195 \pm 229$ | $1583 \pm 665$ | $1582 \pm 549$ |
| GPSample 96D ls=0.1 (within) | $0.93 \pm 0.40$ | $27048 \pm 10433$ | $959 \pm 634$ | – | $28 \pm 4$ | $5902 \pm 3369$ | $3710 \pm 295$ | $21287 \pm 4044$ |
| GPSample 96D ls=0.5 (within) | $1.7 \pm 1.4$ | $11749 \pm 6302$ | $773 \pm 631$ | – | $45 \pm 5$ | $4142 \pm 603$ | $7727 \pm 3790$ | $7131 \pm 2780$ |
| GPSample 8D ls=0.1 (outof) | $0.042 \pm 0.003$ | $40 \pm 6$ | $16 \pm 4$ | $368 \pm 40$ | $3.1 \pm 1.1$ | $58 \pm 4$ | $23 \pm 6$ | $31 \pm 9$ |
| GPSample 8D ls=0.5 (outof) | $0.044 \pm 0.007$ | $42 \pm 8$ | $15 \pm 2$ | $432 \pm 25$ | $2.9 \pm 0.6$ | $56 \pm 8$ | $22 \pm 3$ | $32 \pm 4$ |
| GPSample 16D ls=0.1 (outof) | $0.18 \pm 0.26$ | $420 \pm 673$ | $37 \pm 3$ | $1939 \pm 161$ | $10.0 \pm 18.6$ | $303 \pm 527$ | $309 \pm 759$ | $122 \pm 55$ |
| GPSample 16D ls=0.5 (outof) | $0.25 \pm 0.48$ | $191 \pm 98$ | $35 \pm 3$ | $1984 \pm 242$ | $4.0 \pm 1.8$ | $135 \pm 18$ | $61 \pm 15$ | $101 \pm 17$ |
| GPSample 32D ls=0.1 (outof) | $0.22 \pm 0.06$ | $775 \pm 367$ | $112 \pm 27$ | $13568 \pm 1851$ | $7.0 \pm 0.9$ | $489 \pm 88$ | $310 \pm 54$ | $782 \pm 429$ |
| GPSample 32D ls=0.5 (outof) | $0.20 \pm 0.03$ | $943 \pm 599$ | $107 \pm 45$ | $14496 \pm 3794$ | $8.1 \pm 1.2$ | $445 \pm 82$ | $264 \pm 54$ | $578 \pm 252$ |
| GPSample 64D ls=0.1 (outof) | $0.87 \pm 0.73$ | $3210 \pm 2094$ | $401 \pm 196$ | – | $36 \pm 10$ | $2682 \pm 694$ | $2650 \pm 266$ | $4494 \pm 1290$ |
| GPSample 64D ls=0.5 (outof) | $1.0 \pm 0.8$ | $3873 \pm 1609$ | $456 \pm 178$ | – | $49 \pm 27$ | $2683 \pm 368$ | $2422 \pm 339$ | $3716 \pm 1075$ |
| GPSample 96D ls=0.1 (outof) | $0.93 \pm 0.21$ | $12120 \pm 3617$ | $1425 \pm 691$ | – | $109 \pm 54$ | $11096 \pm 1346$ | $12789 \pm 1605$ | $17105 \pm 2013$ |
| GPSample 96D ls=0.5 (outof) | $0.78 \pm 0.19$ | $16148 \pm 11031$ | $1228 \pm 491$ | – | $100 \pm 22$ | $11050 \pm 2367$ | $11315 \pm 2092$ | $15668 \pm 3832$ |

Table 14: Compute time per algorithm across RL benchmark tasks.

| | Global methods | | | Local methods | | | | |
|---|---|---|---|---|---|---|---|---|
| Task | Sobol | qEUBO ($q = 1$) | LineCoSpar | G2L-GIPBOr | GIPBOr | TuRPBO | PrefSQP | G2L-PrefSQP |
| Hopper | $3.7 \pm 0.4$ | $1794 \pm 1319$ | $163 \pm 57$ | $835 \pm 608$ | $30 \pm 10$ | $803 \pm 629$ | $379 \pm 146$ | $1232 \pm 769$ |
| Walker2d | $12 \pm 1$ | $17951 \pm 6276$ | $2370 \pm 2445$ | $24990 \pm 3968$ | $168 \pm 32$ | $15050 \pm 3613$ | $13518 \pm 2107$ | $17597 \pm 4866$ |

Table 15: Runtime (s) on the Sushi 4D benchmark.

| Algorithm | $n_{\text{init}} = 2$ | Comparisons | | | Anchor | | |
|---|---|---|---|---|---|---|---|
| | Baseline | $1 \times d$ | $2 \times d$ | $4 \times d$ | $1 \times d$ | $2 \times d$ | $4 \times d$ |
| Sobol | $\mathbf{0.02 \pm 0.00}$ | $\mathbf{0.02 \pm 0.00}$ | $\mathbf{0.02 \pm 0.00}$ | $\mathbf{0.02 \pm 0.00}$ | $\mathbf{0.02 \pm 0.00}$ | $\mathbf{0.02 \pm 0.00}$ | $\mathbf{0.02 \pm 0.00}$ |
| GLISp | $1.98 \pm 2.47$ | $0.94 \pm 0.59$ | $1.46 \pm 1.76$ | $3.07 \pm 0.15$ | $2.67 \pm 2.66$ | $2.50 \pm 1.87$ | $1.73 \pm 1.67$ |
| qEUBO | $17.54 \pm 1.44$ | $16.30 \pm 2.41$ | $18.39 \pm 3.13$ | $18.57 \pm 1.95$ | $16.98 \pm 1.88$ | $17.27 \pm 4.23$ | $18.37 \pm 5.88$ |
| PrefSQP | $10.93 \pm 3.43$ | $8.89 \pm 1.94$ | $10.12 \pm 1.83$ | $8.47 \pm 0.76$ | $8.16 \pm 2.15$ | $11.20 \pm 7.41$ | $11.54 \pm 5.46$ |
| LineCoSpar | $6.15 \pm 1.81$ | $6.00 \pm 1.34$ | $7.17 \pm 1.85$ | $6.83 \pm 1.06$ | $6.24 \pm 0.87$ | $7.34 \pm 2.89$ | $7.80 \pm 2.72$ |
| GIPBOr | $1.90 \pm 0.35$ | $2.43 \pm 0.82$ | $2.47 \pm 0.53$ | $2.21 \pm 0.19$ | $2.38 \pm 0.65$ | $2.22 \pm 0.40$ | $3.66 \pm 4.61$ |
| GIPBO | $85.28 \pm 7.14$ | $95.89 \pm 13.05$ | $106.61 \pm 8.94$ | $114.58 \pm 8.07$ | $90.37 \pm 6.61$ | $89.45 \pm 11.33$ | $88.68 \pm 7.98$ |
| G2L GIPBOr | $12.41 \pm 0.94$ | $12.05 \pm 1.34$ | $12.86 \pm 1.84$ | $12.74 \pm 1.53$ | $11.70 \pm 1.44$ | $11.75 \pm 1.38$ | $11.33 \pm 2.21$ |
| G2L GIPBO | $89.66 \pm 10.47$ | $86.95 \pm 15.11$ | $97.24 \pm 11.82$ | $101.57 \pm 14.92$ | $86.94 \pm 16.94$ | $85.12 \pm 15.71$ | $83.01 \pm 11.11$ |
| G2L PrefSQP | $14.11 \pm 2.63$ | $13.65 \pm 1.50$ | $13.13 \pm 2.04$ | $14.19 \pm 1.72$ | $14.01 \pm 3.51$ | $12.85 \pm 1.92$ | $13.92 \pm 2.10$ |
| TuRPBO | $29.07 \pm 2.43$ | $29.81 \pm 2.47$ | $35.36 \pm 21.29$ | $27.51 \pm 2.06$ | $27.87 \pm 2.56$ | $31.37 \pm 4.07$ | $27.93 \pm 3.57$ |

Table 16: Runtime (s) on the CarCab 7D benchmark.

| Algorithm | $n_{\text{init}} = 2$ | Comparisons | | | Anchor | | |
|---|---|---|---|---|---|---|---|
| | Baseline | $1 \times d$ | $2 \times d$ | $4 \times d$ | $1 \times d$ | $2 \times d$ | $4 \times d$ |
| Sobol | $\mathbf{0.08 \pm 0.01}$ | $\mathbf{0.08 \pm 0.01}$ | $\mathbf{0.09 \pm 0.00}$ | $\mathbf{0.09 \pm 0.01}$ | $\mathbf{0.09 \pm 0.01}$ | $\mathbf{0.08 \pm 0.01}$ | $\mathbf{0.08 \pm 0.01}$ |
| GLISp | $7.58 \pm 6.75$ | $8.70 \pm 7.54$ | $7.08 \pm 1.94$ | $8.23 \pm 0.78$ | $10.14 \pm 7.43$ | $8.63 \pm 6.12$ | $9.85 \pm 8.12$ |
| qEUBO | $31.31 \pm 3.06$ | $37.93 \pm 9.65$ | $38.17 \pm 10.39$ | $50.95 \pm 11.36$ | $32.94 \pm 4.44$ | $30.76 \pm 6.51$ | $28.88 \pm 2.91$ |
| PrefSQP | $18.69 \pm 3.34$ | $21.28 \pm 5.74$ | $27.21 \pm 13.60$ | $22.48 \pm 3.42$ | $19.82 \pm 5.59$ | $16.98 \pm 3.28$ | $18.08 \pm 3.01$ |
| LineCoSpar | $11.52 \pm 1.56$ | $13.25 \pm 0.87$ | $13.76 \pm 1.35$ | $17.27 \pm 1.57$ | $13.85 \pm 2.38$ | $12.82 \pm 3.32$ | $12.69 \pm 0.79$ |
| GIPBOr | $2.43 \pm 0.37$ | $3.16 \pm 1.12$ | $3.35 \pm 1.15$ | $4.08 \pm 2.36$ | $3.18 \pm 1.77$ | $2.69 \pm 0.45$ | $2.97 \pm 0.93$ |
| GIPBO | $322.69 \pm 23.14$ | $365.59 \pm 30.47$ | $389.05 \pm 29.87$ | $480.73 \pm 47.32$ | $325.23 \pm 31.72$ | $294.48 \pm 37.33$ | $314.30 \pm 23.15$ |
| G2L GIPBOr | $24.95 \pm 4.21$ | $28.25 \pm 7.38$ | $27.14 \pm 5.72$ | $34.10 \pm 5.65$ | $27.38 \pm 6.29$ | $24.07 \pm 4.74$ | $24.50 \pm 2.11$ |
| G2L GIPBO | $311.65 \pm 27.90$ | $322.22 \pm 31.44$ | $374.62 \pm 46.39$ | $502.68 \pm 114.81$ | $308.21 \pm 30.51$ | $296.53 \pm 47.26$ | $298.86 \pm 30.32$ |
| G2L PrefSQP | $24.08 \pm 3.14$ | $31.94 \pm 7.52$ | $30.02 \pm 5.93$ | $34.80 \pm 6.24$ | $30.23 \pm 6.55$ | $25.32 \pm 3.69$ | $25.76 \pm 4.15$ |
| TuRPBO | $50.47 \pm 2.47$ | $51.53 \pm 3.18$ | $52.49 \pm 3.58$ | $58.18 \pm 2.26$ | $57.78 \pm 17.97$ | $50.03 \pm 5.04$ | $50.58 \pm 2.43$ |

The used software packages are listed in table 17.

Table 17: Software packages used in the benchmark environment.

| **Package** | **Version** |
|---|---|
| Python | 3.12 |
| NumPy | 1.26.4 |
| SciPy | 1.12.0 |
| pandas | 3.0.2 |
| scikit-learn | 1.8.0 |
| numba | 0.65.1 |
| Cython | 3.2.4 |
| Matplotlib | 3.10.9 |
| PyTorch | 2.11.0 (CPU) |
| GPyTorch | 1.14 |
| linear-operator | 0.6 |
| BoTorch | 0.15.1 |
| GPy | 1.13.2 |
| paramz | 0.9.6 |
| CVXOPT | 1.3.3 |
| GLIS | 2.0.2 |
| pyDOE | 0.9.3 |
| pyswarm | 0.8.0 |
| pyro-ppl | 1.9.1 |
| MuJoCo | 3.8.0 |
| Gymnasium | 1.3.0 |
| tqdm | 4.67.3 |