Bayesian Optimization for Noisy Black-box Functions: A Comparative Study Using BoTorch

CHEN Yao
Department of Data Science
City University of Hong Kong
ychen3353-c@my.cityu.edu.hk

CHEN Ziang
Department of Data Science
City University of Hong Kong
ziangchen9-c@my.cityu.edu.hk

Abstract

Bayesian optimization offers an efficient approach for optimizing expensive-to-evaluate noisy black-box functions. Throughout its development, various mean functions, kernel functions, and acquisition functions have been employed to address various Bayesian optimization scenarios, with novel acquisition functions continually emerging. Recent advancements in PyTorch-based libraries like GPyTorch and BoTorch facilitate systematic comparisons of optimization performance across different acquisition functions. In this paper we employ five benchmark test functions (Ackley, Bohachevsky, Booth,Easom, Three-Hump Camel) as objectives to compare optimization effectiveness under combinations of three mean functions (Constant, Linear, Quadratic), three kernel functions (RBF, Matern(3/2), Matern(5/2)), and eight acquisition functions suitable for noisy single-objective optimization (qSR, qUCB, qLogNoisyEI, qPES, qMVE, qLBMVE, qKG, TS). Additionally, we present a BoTorch-based implementation framework for Bayesian optimization with discussions on critical implementation details. The code is available at https://github.com/ZiangChen9/SDSC6002.git.

1 Introduction

In many real-world optimization problems, particularly those characterized by expensive and noisy objective functions, the number of allowable function evaluations is often severely limited due to time, cost, or resource constraints. Traditional optimization methods may struggle in such settings, either requiring an impractically large number of evaluations or being overly sensitive to noise. Bayesian Optimization (BO) has emerged as a powerful framework to address these challenges. By constructing a probabilistic surrogate model of the objective function, BO enables efficient global optimization through informed sampling strategies that balance exploration and exploitation Shahriari et al. [2016].

A typical Bayesian Optimization procedure consists of the following steps:

1. A small set of initial input points is selected—usually via space-filling methods (such as Latin Hypercube Sampling or Sobol sequences) and the corresponding noisy objective function values are observed.
2. A probabilistic surrogate model, most commonly a Gaussian Process (GP), is fitted to the collected data, yielding a posterior distribution over the objective function.
3. An acquisition function is optimized to determine the next query point by leveraging both the predicted mean and uncertainty from the GP model.
4. The true (noisy) objective function is evaluated at the selected point, and the surrogate model is updated with the new observation.

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

5. Steps (2) to (4) are repeated iteratively until the predefined evaluation budget is exhausted.

This model-based optimization process is particularly well-suited for problems where each function evaluation is expensive and prone to stochasticity.

To facilitate the implementation and experimentation of Bayesian Optimization algorithms, the BoTorch library has been introduced as a modern, modular, and flexible platform. Built on top of PyTorch and seamlessly integrated with GPyTorch, BoTorch provides comprehensive support for defining and optimizing a wide range of surrogate models and acquisition functions. Notably, it offers a suite of advanced tools for modeling noisy observations, batch evaluations, and parallel optimization, making it well-suited for high-cost, noisy settings. However, despite its capabilities, BoTorch's current documentation and usage examples remain limited, posing a potential barrier for researchers and practitioners seeking to explore its full potential.

In this paper, we present a systematic empirical study on the performance of eight Monte Carlo-based acquisition functions in noisy single-objective Bayesian optimization. Our evaluation covers 9 Gaussian Process configurations, formed by combining three types of prior mean functions (Constant, Linear, Quadratic) with three commonly used kernel functions (RBF, Matérn($3/2$), Matérn($5/2$)). We benchmark these configurations on five representative test functions—Ackley, Bohachevsky, Booth, Easom, and Three-Hump Camel—chosen to reflect a variety of optimization landscapes including multi-modal, bowl-shaped, plate-shaped, rugged, and valley-shaped structures. The remainder of this paper is organized as follows:

- **Section 2** introduces the Gaussian Process surrogate modeling framework and the acquisition functions used in this paper.
- **Section 3** describes the experimental setup and evaluation protocol.
- **Section 4** presents the empirical results.
- **Section 5** offers a detailed discussion and interpretation of the findings.
- **Section 6** concludes the paper with key takeaways, limitations, and directions for future work.

2 Bayesian Optimization Framework

Bayesian Optimization (BO) is fundamentally built upon two closely coupled components: a probabilistic surrogate model, which approximates the unknown objective function, and an acquisition function, which guides the selection of future query points based on the surrogate's posterior distribution. These two components form the backbone of the BO loop and jointly determine the sample efficiency and robustness of the optimization process.

in this paper, we adopt a **Gaussian Process (GP)** as the surrogate model due to its non-parametric nature and closed-form posterior inference, which make it particularly well-suited for modeling noisy and expensive black-box functions. The GP not only provides point estimates of the objective function but also quantifies predictive uncertainty, enabling a principled trade-off between exploration and exploitation.

The **acquisition function**, defined on top of the GP posterior, evaluates the utility of candidate points by scoring their potential to improve the objective or reduce uncertainty about its optimum. At each iteration, the next evaluation point is selected by maximizing the acquisition function over the input domain, after which the true (noisy) objective value is observed and incorporated into the model. This iterative mechanism gradually refines the surrogate and steers the optimization towards optimal regions.

The overall Bayesian optimization procedure adopted in this paper is summarized in **Figure 1** and detailed in **Algorithm 1**. The process begins with an initial design generated via a space-filling Sobol sequence, followed by repeated model fitting and acquisition optimization steps, until the evaluation budget is exhausted.

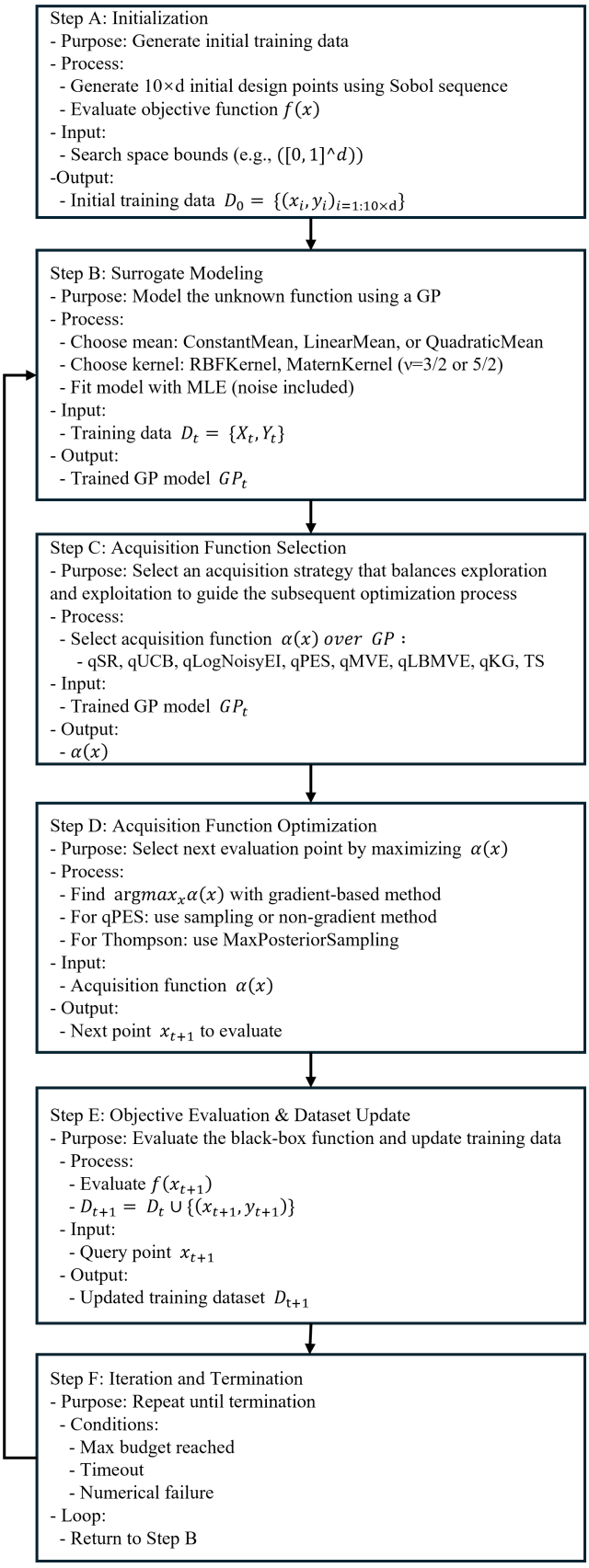


Figure 1: Bayesian Optimization Loop

Algorithm 1 Bayesian Optimization with Gaussian Process Surrogate Model

Input: dimension d, $budget$, objective function $f(\cdot)$, noise ϵ, $y(\cdot) = f(\cdot) + \epsilon$
Output: final dataset $\mathcal{D}$, estimated optimal solution $\mathbf{x}^*$
Initialization:
  Generate $10d$ initial points using Sobol sequence: $\mathbf{X}_0 = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{10d}\}$
  Evaluate the objective function for each initial point: $\mathbf{Y}_0 = \{y(\mathbf{x}_1), y(\mathbf{x}_2) \ldots, y(\mathbf{x}_{10d})\}$
  Initialize the dataset: $\mathcal{D}_0 = \{(\mathbf{x}_i, y_i) \mid i = 1, 2, \ldots, 10d\}$
Construct Gaussian Process Surrogate Model:
  Select mean functions:
    $\mathcal{M} = \{\mu_{\text{Constant}}(\mathbf{x}), \mu_{\text{Linear}}(\mathbf{x}), \mu_{\text{Quadratic}}(\mathbf{x})\}$
  Select covariance functions:
    $\mathcal{K} = \{k_{\text{RBF}}(\mathbf{x}, \mathbf{x}'), k_{\text{Matérn}(3/2)}(\mathbf{x}, \mathbf{x}'), k_{\text{Matérn}(5/2)}(\mathbf{x}, \mathbf{x}')\}$
  Estimate parameters via maximum likelihood estimation on $\mathcal{D}$
  Fit GP surrogate model $GP(\mathcal{D})$
Select Acquisition Functions:
  Acquisition Functions set:
  $\mathcal{A} = \{\alpha_{\text{qSR}}, \alpha_{\text{qUCB}}, \alpha_{\text{qLogNoisyEI}}, \alpha_{\text{qPES}}, \alpha_{\text{qMVE}}, \alpha_{\text{qLBMVE}}, \alpha_{\text{qKG}}, \alpha_{\text{TS}}\}$
while budget T not exhausted **and** not timed out **and** no numerical instability **do**
                                                                ▷ Optimization Process
  Score candidate points using $\alpha(\cdot) \in \mathcal{A}$ based on current $GP(\mathcal{D})$ ▷ Scoring
                                                                ▷ Acquisition function Optimization
    if $\alpha(\cdot) = \alpha_{\text{TS}}$
      See Algorithm 2
    else if$\alpha(\cdot) \neq \alpha_{\text{qPES}}$
      Use gradient-based optimization to find $\mathbf{x}_{\text{new}} = \arg\max \alpha(\mathbf{x})$
    else
      Use gradient-free optimization to find $\mathbf{x}_{\text{new}} = \arg\max \alpha(\mathbf{x})$
    end if
    Evaluate the objective function at the new point: $y_{\mathbf{new}} = f(\mathbf{x}_{\mathbf{new}}) + \epsilon$
    Update the dataset: $\mathcal{D}' = \mathcal{D} \cup \{(\mathbf{x}_{\mathbf{new}}, y_{\mathbf{new}})$
    Refit $GP(\mathcal{D}')$
end while
Output Results:
  Find $\mathbf{x}^* = \arg\min_{(\mathbf{x},y)\in\mathcal{D}} y$
  Return $f(\mathbf{x}^*)$ and $\mathcal{D}$

2.1 Gaussian Process Surrogate Model

in this paper, we employ a **noisy Gaussian Process (GP) surrogate model** for Bayesian optimization. A GP is a nonparametric model defined by a prior mean function $\mu_0(\boldsymbol{x})$ and a positive semi-definite covariance function $k(\boldsymbol{x}, \boldsymbol{x}')$. Given a set of input-output pairs $\mathcal{D}_n = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$, the GP assumes that the latent function values $\boldsymbol{f} = [f(\boldsymbol{x}_1), \ldots, f(\boldsymbol{x}_n)]^\top$ and the observations $\boldsymbol{y} = \boldsymbol{f} + \boldsymbol{\epsilon}$ are jointly Gaussian:

$$\boldsymbol{f}|\boldsymbol{X}_{1:n} \sim \mathcal{N}(\boldsymbol{m}, \boldsymbol{K}), \quad \boldsymbol{y}|\boldsymbol{f} \sim \mathcal{N}(\boldsymbol{f}, \sigma^2\boldsymbol{I}) \tag{1}$$

where:

- $\boldsymbol{X}_{1:n} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ is the set of training input points
- $\boldsymbol{m} = [\mu_0(\boldsymbol{x}_1), \ldots, \mu_0(\boldsymbol{x}_n)]^\top$ is the prior mean vector
- $\boldsymbol{K}$ is the covariance matrix with elements $K_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$
- $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \sigma^2\boldsymbol{I})$ represents the observation noise
- σ^2 is the observation noise variance

The **posterior mean** and **variance** of the function value at a test point $\boldsymbol{x}$ are given by:

$$\mu_n(\boldsymbol{x}) = \mu_0(\boldsymbol{x}) + \boldsymbol{k}(\boldsymbol{x})^\top (\boldsymbol{K} + \sigma^2 \boldsymbol{I})^{-1} (\boldsymbol{y} - \boldsymbol{m}) \tag{2}$$

$$\sigma_n^2(\boldsymbol{x}) = k(\boldsymbol{x}, \boldsymbol{x}) - \boldsymbol{k}(\boldsymbol{x})^\top (\boldsymbol{K} + \sigma^2 \boldsymbol{I})^{-1} \boldsymbol{k}(\boldsymbol{x}) \tag{3}$$

Here, $\boldsymbol{k}(\boldsymbol{x}) = [k(\boldsymbol{x}, \boldsymbol{x}_1), \ldots, k(\boldsymbol{x}, \boldsymbol{x}_n)]^\top$ is the vector of covariances between $\boldsymbol{x}$ and the training inputs.

The **prior mean function** $\mu_0(\boldsymbol{x})$ in our experiments is selected from:

- **Constant mean**: $\mu_0(\boldsymbol{x}) = \mu_0$
- **Linear mean**: $\mu_0(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} + b$
- **Quadratic mean**: $\mu_0(\boldsymbol{x}) = \boldsymbol{x}^\top \boldsymbol{A} \boldsymbol{x} + \boldsymbol{w}^\top \boldsymbol{x} + b$

We also experiment with three different covariance functions:

- **Radial Basis Function (RBF) kernel**:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \theta_0^2 \exp\left(-\frac{1}{2} r^2\right)$$

- **Matérn kernel** ($\nu = 3/2$):

$$k(\boldsymbol{x}, \boldsymbol{x}') = \theta_0^2 \left(1 + \sqrt{3} r\right) \exp(-\sqrt{3} r)$$

- **Matérn kernel** ($\nu = 5/2$):

$$k(\boldsymbol{x}, \boldsymbol{x}') = \theta_0^2 \left(1 + \sqrt{5} r + \frac{5}{3} r^2\right) \exp(-\sqrt{5} r)$$

with $r^2 = (\boldsymbol{x} - \boldsymbol{x}')^\top \boldsymbol{\Lambda} (\boldsymbol{x} - \boldsymbol{x}')$, and $\boldsymbol{\Lambda}$ is a diagonal matrix of inverse squared length-scales.

In these covariance functions:

- The hyperparameter θ_0^2 represents the signal variance
- $\boldsymbol{\Lambda} = \mathrm{diag}(\lambda_1^{-2}, \ldots, \lambda_d^{-2})$ contains length-scale parameters

To learn the hyperparameters $\boldsymbol{\theta} = (\theta_0, \mu_0, \sigma^2, \lambda_1, \ldots, \lambda_d)$, we maximize the log marginal likelihood:

$$\log p(\boldsymbol{y} | \boldsymbol{X}_{1:n}, \boldsymbol{\theta}) = -\frac{1}{2} (\boldsymbol{y} - \boldsymbol{m})^\top (\boldsymbol{K} + \sigma^2 \boldsymbol{I})^{-1} (\boldsymbol{y} - \boldsymbol{m}) - \frac{1}{2} \log |\boldsymbol{K} + \sigma^2 \boldsymbol{I}| - \frac{n}{2} \log(2\pi) \tag{4}$$

2.2 Acquisition Functions in Bayesian Optimization

In Bayesian Optimization (BO), acquisition functions are critical for directing the search towards the global optimum of a black-box function. Given a probabilistic surrogate model—typically a Gaussian Process (GP)—the acquisition function evaluates candidate points based on their potential to improve upon the current best observation. It effectively balances *exploration* of uncertain regions and *exploitation* of areas with promising predictions. At each iteration, the point that maximizes the acquisition function is selected for evaluation, thereby driving the optimization process.

This study employs the BoTorch library, a flexible and modular framework built on PyTorch, to implement the BO process. BoTorch offers both **analytic** acquisition functions, which have closed-form expressions suitable for single-point evaluations (i.e., $q = 1$), and **(quasi-) Monte Carlo (MC)** acquisition functions, which approximate expectations via sampling and can handle more complex scenarios, such as batch evaluations ($q > 1$) and noisy observations. Notably, MC-based acquisition functions in BoTorch typically begin with the prefix q, such as *qExpectedImprovement* or *qUpperConfidenceBound*, indicating support for evaluating multiple candidates simultaneously.

Although MC acquisition functions are often associated with batch evaluations, they are equally applicable and beneficial in single-point settings, especially when dealing with noisy objective functions. Analytic acquisition functions generally assume noise-free outputs, which can lead to biased decisions in the presence of stochasticity. In contrast, MC-based methods estimate acquisition values by integrating over the posterior distribution of the surrogate model, thereby capturing uncertainty introduced by observational noise. Furthermore, BoTorch enhances efficiency by leveraging techniques such as the **reparameterization trick**, **(quasi-) Monte Carlo sampling**, and **fixed base samples**, which allow the acquisition functions to be optimized using deterministic and fast gradient-based methods like L-BFGS. These advantages make MC-based acquisition functions not only more robust under noise but also computationally efficient in practice.

Consequently, this paper focuses on MC-based acquisition functions within the BoTorch framework for optimizing noisy single-objective functions using GP surrogates. This choice leverages their robustness to noise and computational efficiency, even in the absence of batch evaluations.

2.2.1 qSimpleRegret(qSR)

The qSimpleRegret (qSR) acquisition function evaluates the expected immediate reward of evaluating a candidate point set $X = (x_1, \ldots, x_q)$ by computing the expectation of the maximum posterior prediction from the Gaussian process (GP) surrogate model. Formally, it is defined as:

$$\alpha_{qSR}(X) = \mathbb{E}[\max Y], \quad Y \sim GP(X),$$

where Y represents posterior samples at X. Unlike improvement-based methods, qSR directly targets the *current best predicted value* rather than potential future gains, making it particularly suitable for the final recommendation phase in Bayesian optimization (BO) where the goal is to identify the optimal observed solution Balandat et al. [2020].

In BoTorch, qSR is implemented through quasi-Monte Carlo (QMC) sampling (e.g., `SobolQMCNormalSampler`) to approximate the expectation. The algorithm generates low-discrepancy samples from the GP posterior, computes the maximum value within each sample batch, and averages these maxima to estimate the simple regret. For constrained optimization, BoTorch enforces feasibility via `ConstrainedMCObjective`, which adjusts negative objective values to ensure non-negativity. In practice, the acquisition function is optimized using gradient-based methods (e.g., L-BFGS), with critical parameters including the sampler's sample size (e.g., 1,024 samples) and the number of optimization restarts (e.g., 20) to mitigate local optima. This approach balances computational efficiency with accurate regret estimation, aligning with scenarios requiring reliable final recommendations in noisy environments.

2.2.2 qUpperConfidenceBound(qUCB)

The `qUpperConfidenceBound` (qUCB) acquisition function extends the analytic Upper Confidence Bound (UCB) strategy to noisy and batch optimization settings via Monte Carlo (MC) sampling. Formally, it computes an optimistic bound on the objective function by evaluating the expectation:

$$\alpha_{qUCB}(X) = \mathbb{E}\left[\max\left(\mu + \left|\tilde{Y} - \mu\right|\right)\right], \quad \tilde{Y} \sim \mathcal{N}\left(\mu, \beta\frac{\pi}{2}\Sigma\right),$$

where μ and Σ are the posterior mean and covariance of the Gaussian process (GP) surrogate model at candidate points X, and β is a tunable hyperparameter controlling the exploration-exploitation trade-off. This reparameterization-based formulation, derived from Wilson et al. [2017], ensures theoretical guarantees for regret minimization while accommodating noisy evaluations.

In BoTorch, qUCB is implemented using quasi-Monte Carlo (QMC) sampling (e.g., `SobolQMCNoMALSampler`) to approximate the expectation. The algorithm generates perturbed samples $\tilde{Y}$ from the GP posterior, scales deviations by β, and computes the maximum optimistic value for each sample. For constrained optimization, feasibility is enforced via `ConstrainedMCObjective`, adjusting negative values to ensure numerical stability. In practice, the acquisition function is optimized with gradient-based methods (e.g., L-BFGS), leveraging parameters such as the sampler's sample size (e.g., 1024 samples) and exploration weight β (e.g., $\beta = 0.1$) to prioritize exploitation in noisy single-point evaluations ($q = 1$). This approach balances computational efficiency with robust exploration, making qUCB suitable for scenarios requiring systematic trade-offs between model uncertainty and predicted performance.

2.2.3 qLogNoisyExpectedImprovement(qLogNoisyEI)

The `qLogNoisyExpectedImprovement` (qLogNoisyEI) acquisition function is a Monte Carlo-based method tailored for Bayesian optimization (BO) in noisy settings. Formally, it computes the logarithm of the expected improvement (EI) over a baseline dataset X_{baseline}, defined as:

$$\alpha_{qLogNoisyEI}(X) = \log \mathbb{E}\left[\max\left(\max Y - \max Y_{\text{baseline}}, 0\right)\right],$$

where Y and Y_{baseline} are posterior samples from the Gaussian process (GP) surrogate model at candidate points X and historical evaluations X_{baseline}, respectively. The logarithmic transformation addresses numerical instability in regions with near-zero improvement probabilities, ensuring robust gradient computation during optimization. Unlike analytic EI variants (e.g., ExpectedImprovement), which assume noise-free observations, qLogNoisyEI integrates over perturbed posterior samples ("fantasy" sampling) to explicitly model observational noise, making it suitable for stochastic objective functions.

In BoTorch, qLogNoisyEI is implemented via quasi-Monte Carlo (QMC) sampling, leveraging a SobolQMCNormalSampler to generate low-discrepancy samples from the GP posterior. Key parameters include X_{baseline} (historical data for improvement benchmarking) and the sampler's sample size (e.g., 1,024 samples), which balance computational cost and approximation accuracy. For single-point evaluations ($q = 1$), the acquisition function is optimized using gradient-based methods (e.g., L-BFGS), enabled by reparameterization and fixed base samples to transform stochasticity into deterministic gradients. This design ensures both noise robustness and computational efficiency, aligning with the requirements of optimizing costly-to-evaluate noisy functions.

2.2.4 qPredictiveEntropySearch (qPES)

The `qPredictiveEntropySearch` (qPES) acquisition function implements the Predictive Entropy Search (PES) strategy, which selects candidate points by maximizing the mutual information between the unknown optimizer and the function values observed at candidate locations. This acquisition function estimates the expected reduction in entropy over the distribution of the optimizer after observing new evaluations. It is particularly well-suited for single-objective problems where the goal is to reduce uncertainty about the location of the global minimizer. The mutual information is approximated using an expectation propagation (EP) algorithm, which allows for a tractable estimation of the conditional entropy without requiring analytic posterior computations.

In BoTorch, qPES is implemented for single-outcome Gaussian process (GP) models and requires a predefined set of sampled optimal inputs, which represent plausible global optima drawn from the posterior distribution. The acquisition value is then computed based on the expected information gain from observing function values at the candidate points. Due to the non-differentiable nature of the EP-based approximation, BoTorch recommends using finite-difference optimization techniques. Additionally, since the EP algorithm can be numerically unstable in rare cases, particularly with many EP factors, jitter terms are introduced during training and testing to mitigate such risks. In practice, qPES can yield negative acquisition values due to estimation variance, though this does not impact the correctness of the optimization process. The implementation used in this paper follows BoTorch's recommendations and applies finite-difference optimization (`with_grad=False`) with typical settings such as 20 restarts and 50 raw samples. This approach provides a robust and principled strategy for information-driven search under uncertainty.

2.2.5 qMaxValueEntropySearch (qMVE)

The qMaxValueEntropy (qMVE) acquisition function implements the Max-value Entropy Search (MES) strategy, which selects candidate points by maximizing the mutual information between the unknown global maximum value y_* and the next evaluation point x. Following Wang and Jegelka [2017], the acquisition function is defined as:

$$\alpha_{qMVE}(x) = I(x, y); y_* \mid D_t) = H(p(y \mid D_t, x)) - \mathbb{E}_{y_*}[H(p(y \mid D_t, x, y_*))], \tag{5}$$

where D_t denotes the historical observations up to iteration t, $H(\cdot)$ is the entropy, and $y_* = \max_{x \in \mathcal{X}} f(x)$. This formulation quantifies the reduction in uncertainty about y_* after evaluating x,

prioritizing points that maximally resolve ambiguity in the global maximum value Wang and Jegelka [2017].

In BoTorch, qMVE approximates the expectation through Monte Carlo (MC) sampling over a predefined candidate set $\mathcal{C} \subset \mathcal{X}$, which discretizes the domain to estimate the distribution of y_*. The implementation requires a dense candidate set (e.g., 10,000 points generated via Sobol sequences scaled to the feasible bounds. Posterior samples from the GP are used to compute entropy reduction via kernel density estimation (KDE). For single-point evaluations ($q = 1$), the acquisition function is optimized using gradient-based methods, with critical parameters including the candidate set size (e.g., $|\mathcal{C}| = 10,000$) and the number of optimization restarts (e.g., 20) to avoid local optima. This approach balances computational tractability with accurate entropy estimation, making qMES particularly effective in scenarios where resolving uncertainty about the global optimum is critical, such as noisy or multi-modal optimization tasks.

2.2.6 qLowerBoundMaxValueEntropy (qLBMVE)

The qLowerBoundMaxValueEntropy (qLBMVE) acquisition function, proposed in the General-purpose Information-Based Bayesian Optimization (GIBBON) framework Moss et al. [2021], offers a computationally efficient approximation of the mutual information between the unknown global maximum value and candidate input points. Rather than evaluating the full mutual information, which is often computationally prohibitive, qLB-MES computes a lower bound on this quantity by estimating the information gain through discrete maximum value sampling. This formulation retains the core entropy reduction objective of Max-value Entropy Search (MES) while significantly reducing computational complexity by discretizing the distribution of potential maxima using a predefined candidate set. For a full derivation and explanation of the mathematical formulation, readers are referred to Moss et al. [2021].

In BoTorch, qLBMVE is implemented using quasi-Monte Carlo (QMC) sampling over a dense candidate set—for instance, 10,000 Sobol samples scaled to match the input bounds. These samples are used both to approximate the distribution of the maximum value and to support posterior sampling necessary for entropy estimation via kernel density methods. In the case of single-point selection ($q = 1$), the acquisition function is typically optimized using gradient-based methods such as L-BFGS. Implementation parameters such as `candidate_set=10_000` ensure high-resolution entropy approximation, while settings like `num_restarts=20` help avoid poor local optima. This approach balances computational efficiency with principled exploration, making qLB-MES particularly well-suited for noisy optimization problems with limited evaluation budgets.

2.2.7 qKnowledgeGradient (qKG)

The `qKnowledgeGradient` (qKG) acquisition function implements the batch Knowledge Gradient strategy using a one-shot formulation. It selects candidate points by maximizing the expected utility gain after obtaining hypothetical observations at those points, thereby quantifying the value of information associated with evaluating a batch. This approach approximates the nested expectation over future outcomes by using a set of "fantasy" models—posterior samples drawn under simulated observations. The acquisition function evaluates how these fantasies affect the choice of future optima, either through the model posterior mean or Monte Carlo sampling. This formulation enables decision-theoretic optimization under uncertainty and generalizes to batch settings through joint optimization of candidates and fantasy decisions. A complete derivation of the one-shot qKG formulation is provided in Balandat et al. [2019], with earlier foundations in Frazier et al. [2008] and Wu and Frazier [2016].

In BoTorch, qKG is implemented using fantasy modeling for the outer expectation and allows differentiable or non-differentiable inner expectations. The user specifies the number of fantasy samples (e.g., 128), and all inputs—including real candidates and fantasy designs—are optimized jointly in a single acquisition call. in this paper, qKG is applied in the single-point setting ($q = 1$), with the acquisition function optimized using 20 restarts and 50 raw samples. This configuration leverages the one-shot optimization framework with 128 fantasies, specified as `num_fantasies=128`. The resulting acquisition function provides a scalable approximation to the value of information, making qKG particularly effective for batch decision-making under noisy and high-cost evaluation conditions.

2.2.8 Thompson Sampling(TS)

The Thompson Sampling (TS) strategy is a randomized acquisition approach widely used in Bayesian optimization. Rather than computing acquisition scores for each candidate point, TS draws a single sample function from the posterior distribution of the surrogate model–typically a Gaussian process (GP)–and selects the input that maximizes this sample. Formally, TS replaces the unknown objective function with a single realization drawn from the GP posterior and queries the point where this sample attains its maximum. This sampling-based strategy inherently balances exploration and exploitation: regions with high predictive mean are likely to yield high values in the sample, while regions with high uncertainty have greater variance and thus a higher probability of being selected. Over multiple iterations, this randomness leads to robust and efficient exploration of the search space THOMPSON [1933].

In BoTorch, TS is implemented via the `MaxPosteriorSampling` class, which samples from the multivariate posterior over a predefined candidate set and returns the point with the highest sampled value. Since a sample from the GP posterior can only be defined over a finite number of points, TS requires a dense candidate set spanning the search space. This is typically generated using a Sobol sequence, which offers better coverage than uniform sampling. in this paper, a set of 5,000 Sobol points was generated and rescaled to match the input bounds. Sampling was performed without replacement to ensure diversity, and the selected point–corresponding to the maximum value in the posterior sample–was appended to the current dataset. This approach avoids explicit acquisition score computation and remains both simple and computationally efficient, particularly in noisy or high-dimensional settings. This strategy is simple, computationally efficient, and highly effective in noisy or high-dimensional optimization settings, where classical acquisition functions may become unstable or overly deterministic.

The Thompson sampling approach differs from other seven acquisition functions studied in this paper in that it does not optimize the acquisition function when selecting the next sampling point. Instead, it samples from the predictive distribution of the Gaussian process surrogate model to probabilistically select the next point, thereby balancing exploration and exploitation in a stochastic manner. The Bayesian optimization process using Thompson sampling is detailed in Algorithm 2.

Algorithm 2 Thompson Sampling for Bayesian Optimization

Input: Gaussian Process surrogate model $GP(\mathcal{D})$
Output: Next sampling point $\mathbf{x}_{\text{new}}$
Initialization:
    Define the number of candidate points $N_{\text{candidates}}$
    Generate a set of candidate points $\mathbf{X}_{\text{candidates}} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{N_{\text{candidates}}}\}$
Sampling Process:
for each candidate point $\mathbf{x}_i \in \mathbf{X}_{\text{candidates}}$ **do**
        Predict the mean $\mu(\mathbf{x}_i)$ and variance $\sigma^2(\mathbf{x}_i)$ using $GP(\mathcal{D})$
        Get a sample from the predictive distribution: $\mathcal{N}(\mu(\mathbf{x}_i), \sigma^2(\mathbf{x}_i))$
end for
Selection:
    Select the candidate point with the highest sampled value: $\mathbf{x}_{\text{new}} = \arg\max_{\mathbf{x}_i \in \mathbf{X}_{\text{candidates}}} y_i$
Output:
    Return the selected point $\mathbf{x}_{\text{new}}$ as the next sampling point

3 Experiment

3.1 General Setup

All experiments were conducted on an Ubuntu server equipped with an NVIDIA L40 GPU (40GB-VRAM), with CUDA acceleration enabled to support parallel computations in PyTorch. Detailed specifications of the experimental environment are provided in Table 1. For each optimization task, we began with an initial design of 20 points sampled using a Sobol sequence. Subsequently, 80 Bayesian optimization iterations were performed, resulting in a total evaluation budget of 80 function evaluations beyond the initial design. To ensure statistically reliable results, each optimization

procedure was independently repeated 30 times, referred to as 30 replications. To maintain numerical precision and stability in Bayesian optimization, all computations were performed using float64 (double precision). The implications of using float32 (single precision) will be analyzed in Appendix C. In standard experimental configurations, we set the noise level σ to 0.5.

Table 1: Experimental Environment Configuration

	Configuration
OS	Ubuntu 22.04
Language	Python 3.12.9
Framework	Botorch 0.13.0
CPU	AMD EPYC 9654
GPU	NVIDIA L40(40G)

3.2 Special configuration

To accommodate the unique characteristics of certain benchmark functions and acquisition strategies, we implemented specific experimental configurations as detailed below.

The Easom function was assigned a lower noise level ($\sigma = 0.2$) than other benchmarks ($\sigma = 0.5$) due to its steep, narrow parabolic basin and sharp global optimum. This adjustment ensures a more appropriate signal-to-noise ratio, facilitating effective optimization.

The Bohachevsky and Booth functions were allocated only 10 iterations per replication because their shallow, parabolic landscapes (with gradient norms below 0.1 across 95% of the domain) allow reliable convergence within minimal iterations. This contrasts with multimodal functions like Ackley requiring 80 evaluations to escape local minima.

Regarding the qPES acquisition function, we disabled gradient computation (with `grad=False`) during optimization due to frequent numerical instabilities in its gradient calculation. These instabilities arise from the entropy-based objective's non-smoothness at sample boundaries and the Monte Carlo approximations of the posterior Hessian matrix, which can lead to ill-conditioned gradients.

The candidate set sizes were configured with 10,000 points for qMVE and qLBMVE, compared to 5,000 for qTS. This configuration was determined based on preliminary experiments, which indicated that for qMVE and qLBMVE, a candidate set size below 1,000 was insufficient to achieve satisfactory optimization performance. Increasing the candidate set size to 10,000 significantly improved the maximization of the acquisition function, while the increase in runtime was negligible. For qTS, a candidate set size of 5,000 was chosen to balance computational efficiency and optimization performance. The impact of candidate set size on optimization performance will be further analyzed in Appendix B.

These configurations are summarized in Table 2 and Table 3.

Table 2: Configuration of Benchmark Functions

Function	**Noise Level**	**Repl.**	**Budget (Evaluations)**	**Searching Space (Optimal Value)**
Ackley	0.5	30	80	$[-32.768, 32.768]^2$ (0.0)
Bohachevsky	0.5	30	10	$[-100.0, 100.0]^2$ (0.0)
Booth	0.5	30	10	$[-10.0, 10.0]^2$ (0.0)
Three-Hump Camel	0.5	30	80	$[-5.0, 5.0]^2$ (0.0)
Easom	0.2	30	80	$[-5.0, 5.0]^2$ (-1.0)

3.3 Evaluation

We conducted a comprehensive evaluation of eight acquisition functions across $3 \times 3 = 9$ distinct Gaussian process configurations, representing all combinations of three kernel functions (RBF,

Table 3: Acquisition Function Configuration

Acquisition Function	with gradient	candidate set size	number of initial points
qSR	True	/	20
qUBC	True	/	20
qLogNoisyEI	True	/	20
qPES	False	/	20
qMVE	True	10000	20
qLBMVE	True	10000	20
qKG	True	/	20
TS	/	5000	20

Matérn-1.5, and Matérn-2.5) and three mean functions (constant, linear, and quadratic). This systematic assessment was performed on five benchmark objective functions, resulting in $8 \times 9 \times 5 = 360$ distinct experimental conditions.

This extensive evaluation protocol supports statistically reliable conclusions by incorporating:

- Full factorial design covering major GP modeling choices
- Multiple random initializations (30 replications per condition)

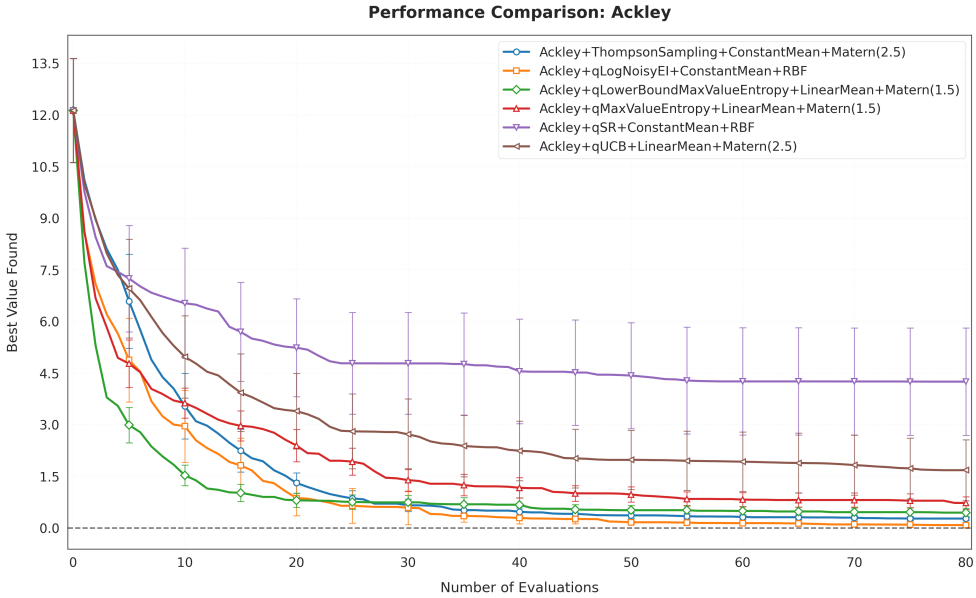
4 Results

To evaluate the performance of different acquisition strategies under noisy single-objective Bayesian optimization, we conducted systematic experiments across five benchmark functions: Ackley, Bohachevsky, Booth, Three-Hump Camel, and Easom. For each benchmark, eight acquisition functions were tested across 9 distinct Gaussian process (GP) configurations, defined by all combinations of three mean functions (constant, linear, quadratic) and three covariance kernels (RBF, Matérn-1.5, Matérn-2.5). Each configuration was evaluated over 30 replications, and the average best value found per evaluation was recorded. Error bars denote 95% confidence intervals calculated using the t-distribution.

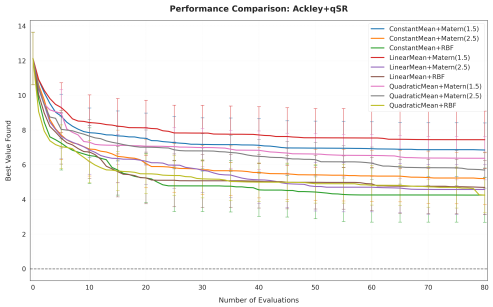
For each acquisition function, performance curves corresponding to the 9 GP configurations were plotted (subfigures b–i). The configuration yielding the best overall performance was selected to represent that acquisition function. These best-performing curves across all acquisition functions were then compared in a summary plot (subfigure a), enabling high-level comparison among acquisition functions for each benchmark. The curve of the "overall" image is derived from the optimal combination of each acquisition function below. For example, in the case of Ackley's qLogNoisyEI, the combination of the constant mean and RBF kernel performs the best, so we select this combination as representation of qLogNoisyEI and incorporate it into the corresponding overall image for Ackley.

Due to occasional numerical instabilities or excessive runtime, not all acquisition-configuration combinations completed successfully. Consequently, the number of plotted curves may differ across figures. The table in last page (Table A1) presents the mean and standard deviation (calculated over 30 replications) of the best observed values at 25%, 50%, 75%, and 100% of the optimization budget (additional evaluations after initial sampling). The results cover 5 test functions evaluated with 8 acquisition functions, where each acquisition function was combined with 3 mean functions and 3 covariance functions (yielding 9 variants per acquisition function).

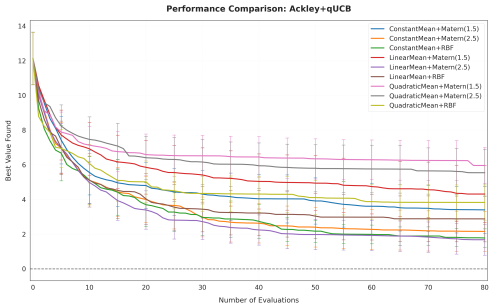
Figure 2: Ackley



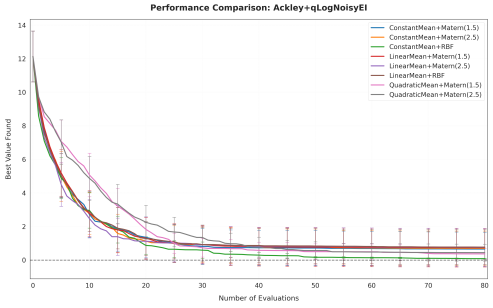
(a) overall



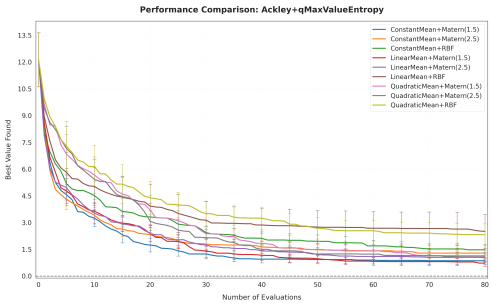
(b) qSR



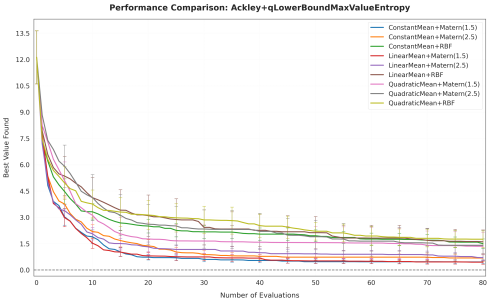
(c) qUCB



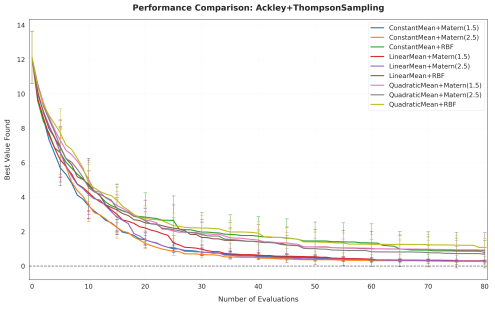
(d) qLogNoisyEI



(e) qMVE

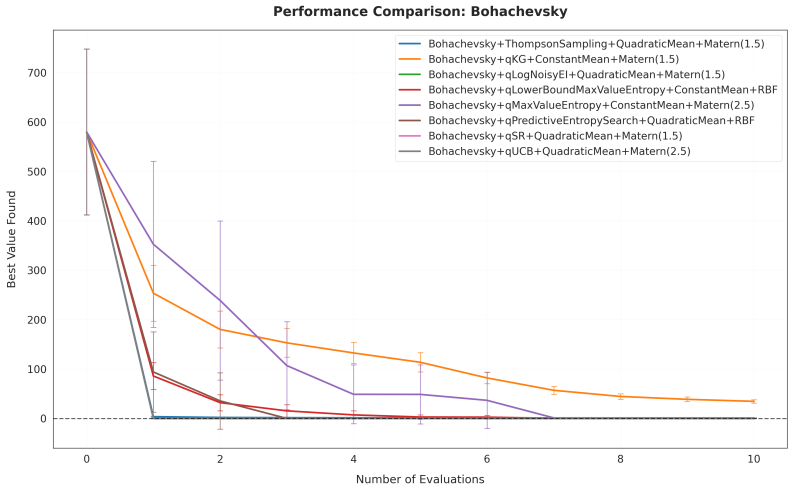


(f) qLBMVE

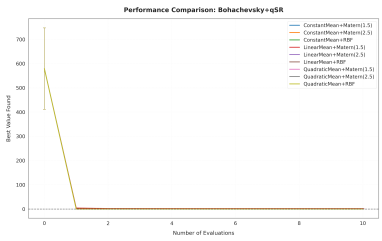


(g) TS

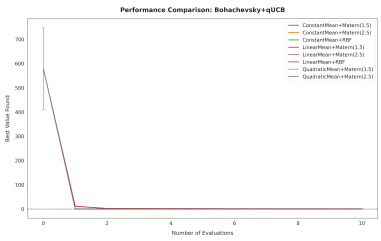
Figure 3: Bohachevsky



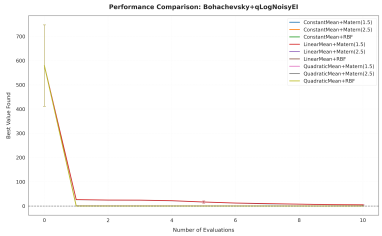
(a) overall



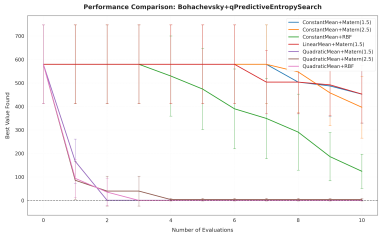
(b) qSR



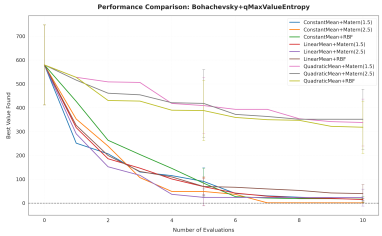
(c) qUCB



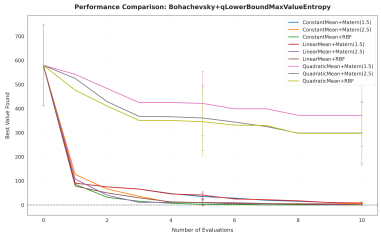
(d) qLogNoisyEI



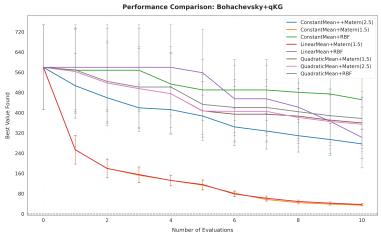
(e) qPES



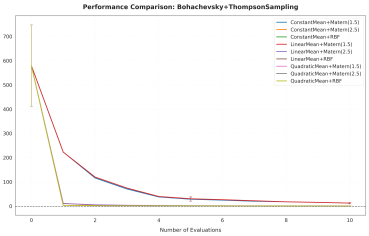
(f) qMVE



(g) qLBMVE

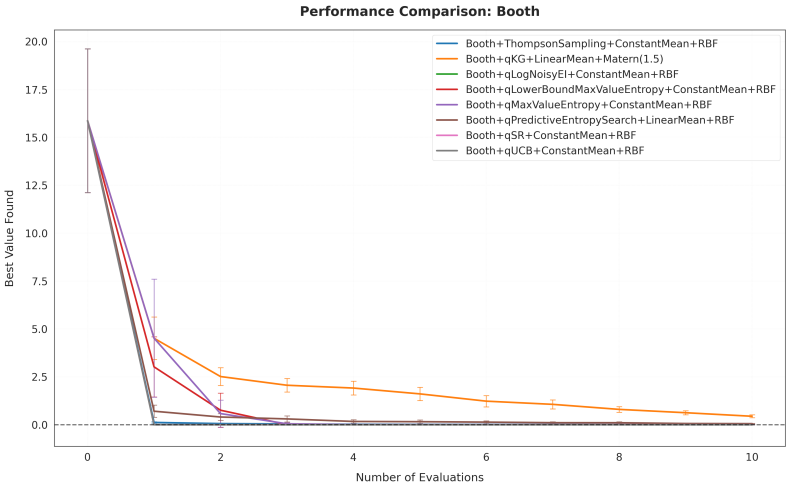


(h) qkG

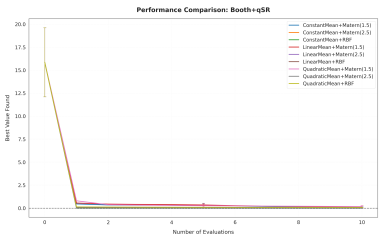


(i) TS

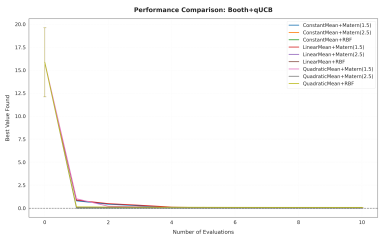
Figure 4: Booth



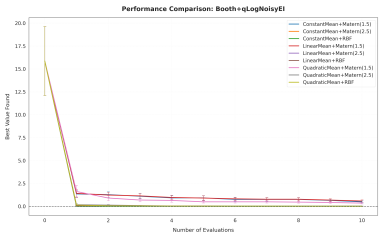
(a) overall



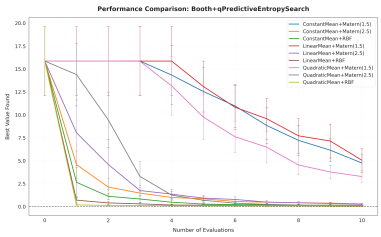
(b) qSR



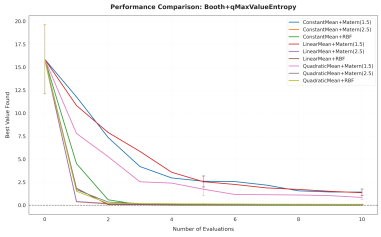
(c) qUCB



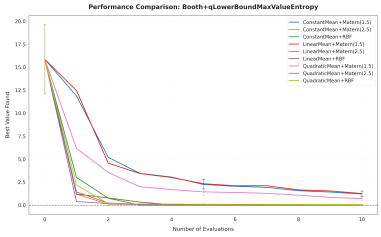
(d) qLogNoisyEI



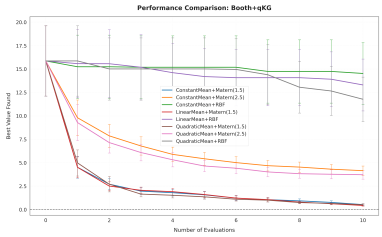
(e) qPES



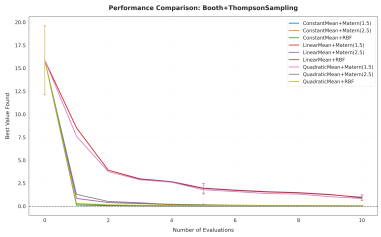
(f) qMVE



(g) qLBMVE

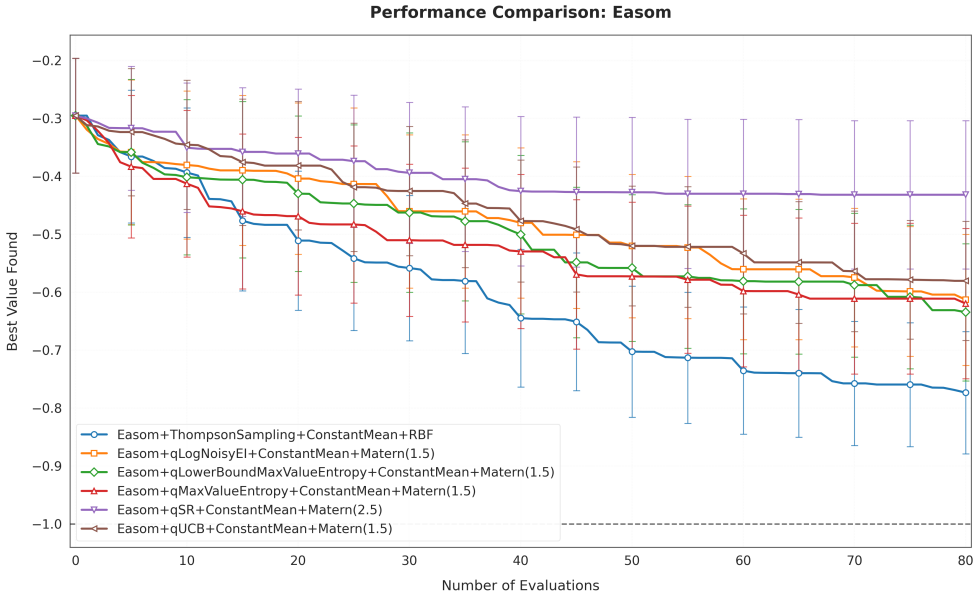


(h) qKG

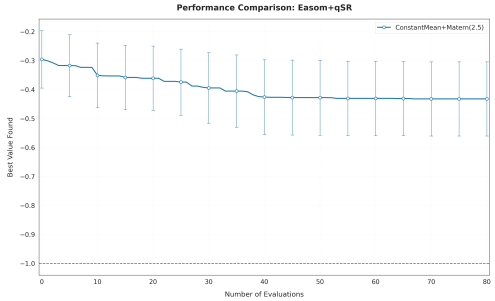


(i) TS

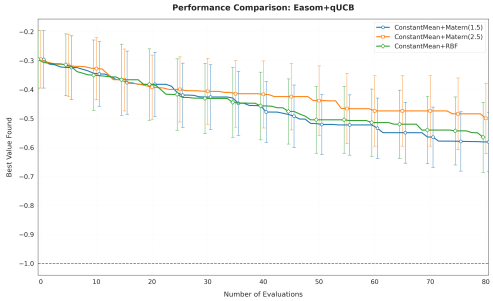
Figure 5: Easom



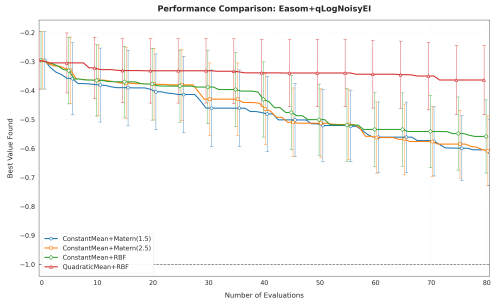
(a) overall



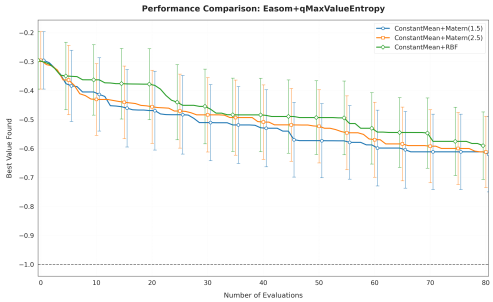
(b) qSR



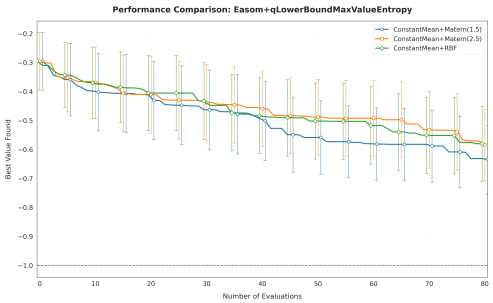
(c) qUCB



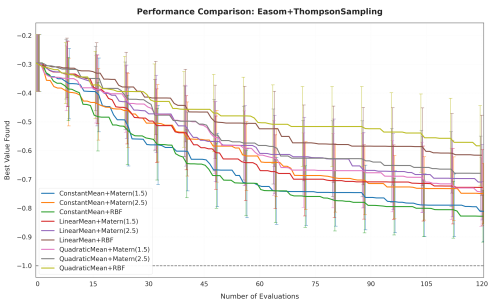
(d) qLogNoisyEI



(e) qMVE

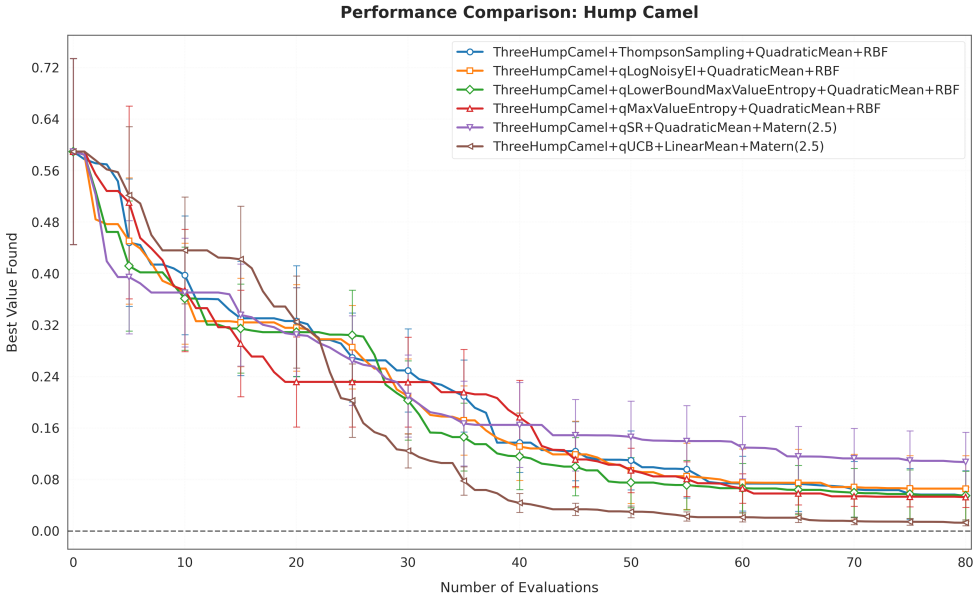


(f) qLBMVE

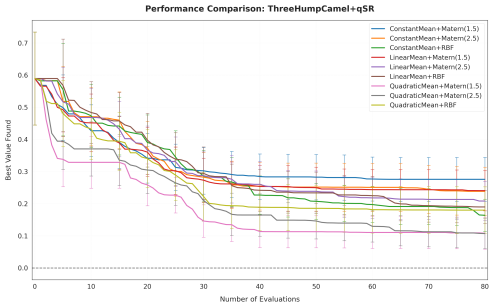


(g) TS

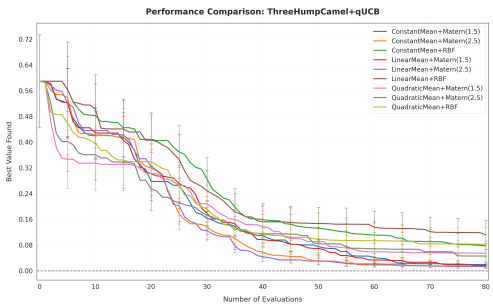
Figure 6: ThreeHumpCamel



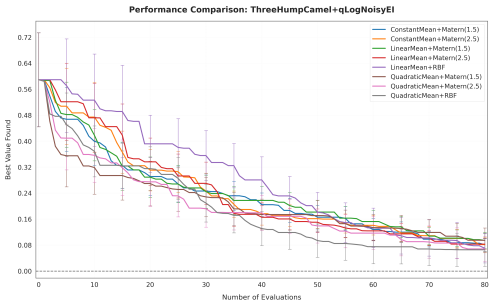
(a) overall



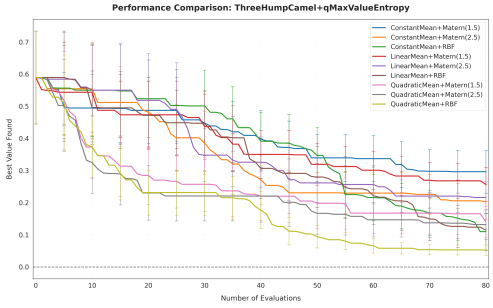
(b) qSR



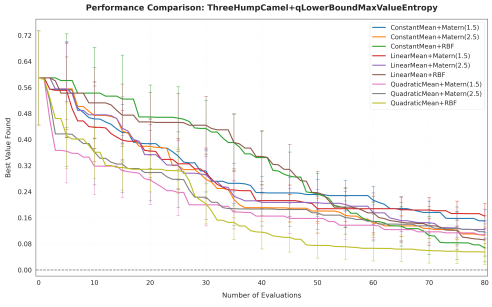
(c) qUCB



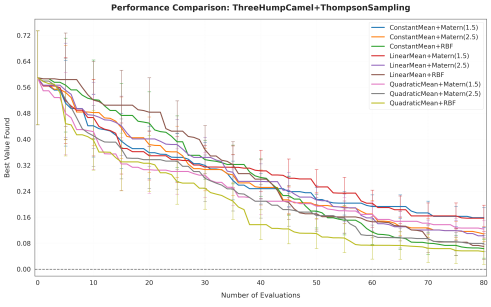
(d) qLogNoisyEI



(e) qMVE



(f) qLBMVE



(g) TS

5 Discussion

This study evaluates the performance of various acquisition functions across several benchmark optimization problems, including the Ackley, Bohachevsky, Booth, Easom, and Three-Hump Camel functions. The focus is on understanding how different acquisition strategies perform under varying levels of function complexity and computational constraints.

Ackley Function

The Ackley function, characterized by its multimodal nature and numerous local optima, poses a significant challenge for optimization algorithms. Among the tested acquisition strategies, qLogNoisyEI, Thompson Sampling (TS), and qLowerBoundMaxValueEntropy (qLBMVE) demonstrated superior performance, consistently converging to low objective values. These methods effectively balance exploration and exploitation, making them suitable for complex, noisy landscapes like Ackley.

In contrast, qSR and qUCB exhibited slower convergence and higher variance, indicating less effective search capabilities in such intricate landscapes. qMaxValueEntropy (qMVE) showed moderate performance, outperforming qSR and qUCB but lagging behind the top-performing methods.

Bohachevsky and Booth Functions

The Bohachevsky and Booth functions are characterized by simple, convex landscapes. Most acquisition functions achieved near-optimal performance within a limited evaluation budget, with minor differences observed among them. This rapid convergence suggests that optimization tasks on these functions are relatively straightforward and less sensitive to the choice of acquisition function.

However, for the Bohachevsky function, qKG, qMaxValueEntropy (qMVE), qPredictiveEntropySearch (qPES), and qLowerBoundMaxValueEntropy (qLBMVE), demonstrated relatively poorer performance compared to qLogNoisyEI, TS, qUCB, and qSR. These underperforming methods often rely on complex sampling or entropy-based approximations, which may not be necessary for simple landscapes and can introduce unnecessary computational overhead.

Easom Function

The Easom function is known for its extremely narrow basin of attraction and a sharp global optimum, making it a particularly difficult target for optimization algorithms. Across all acquisition strategies, most configurations performed poorly, with many failing to produce results. This widespread failure is especially apparent for qSR, likely due to its deterministic and greedy behavior, which fails to explore sufficiently in such a deceptive landscape.

In contrast, Thompson Sampling (TS) emerged as the best-performing acquisition strategy on the Easom function. TS consistently converged closer to the global optimum across all configurations and notably did not fail in any instance. Its stochastic nature enables broader exploration and greater robustness to the function's steep and irregular curvature.

Three-Hump Camel Function

The Three-Hump Camel function presents a relatively smooth and low-dimensional landscape with moderate complexity. Among all acquisition strategies, qUCB achieved the best performance, consistently converging to near-optimal values faster than others across different configurations. In contrast, qSR exhibited the worst performance, with slow convergence and high variance. The remaining acquisition functions—qLogNoisyEI, qLBMVE, qMVE, and TS—demonstrated comparable performance, showing steady convergence and relatively low variance.

It is important to note that both qKG and qPES were excluded from the comparison on the Ackley, Easom, and Three-Hump Camel functions due to excessive computational costs. These methods failed to return results within the allotted time budget, highlighting their scalability limitations under complex or high-dimensional function evaluations.

Summary

1. **Overall Performance of Acquisition Functions** The performance of acquisition functions varies depending on the complexity and structure of the objective function. For the Ackley function, which is highly multimodal, acquisition functions like qLogNoisyEI, TS, and qLBMVE achieved better results by effectively balancing exploration and exploitation. In contrast, the Easom function, which has a narrow global optimum and steep curvature, proved challenging for most methods, but Thompson Sampling consistently performed well due to its stochastic nature. For simple convex functions such as Bohachevsky and Booth, most acquisition functions worked adequately, and complex strategies like qPES or qKG offered limited advantage. On moderately complex functions like the Three-Hump Camel, qUCB showed strong performance, while other methods like TS also remained competitive.
    - For complex multimodal functions like Ackley, which contain many local optima and require balancing exploration and exploitation, strategies such as qLogNoisyEI, TS, and qLBMVE demonstrated strong performance by effectively navigating the complex landscape.
    - For functions like Easom, which are also complex but characterized by a narrow global optimum and steep surrounding curvature, many acquisition functions encountered numerical issues and failed to return valid results. In contrast, Thompson Sampling showed clear advantages—its stochastic nature enables broader exploration and made it more robust and stable under such difficult conditions.
    - For simple convex functions (e.g., Bohachevsky, Booth), most acquisition functions performed adequately. In such cases, using computationally expensive strategies like qKG or qPES is often unnecessary.
    - For moderately complex functions (e.g., Three-Hump Camel), qUCB achieved the best performance, while other acquisition functions, such as TS, also showed competitive results.
2. **Limitations of qKG and qPES**
   qKG and qPES were only included in the comparison for the Bohachevsky and Booth functions due to their high computational costs. Even in these simpler scenarios, their performance was suboptimal compared to other acquisition functions. Their reliance on complex sampling or entropy-based approximations makes them less practical for functions where such sophistication is unnecessary.
3. **Hypotheses regarding the prior mean function**
   Our results suggest that the effectiveness of the prior mean function depends on the shape of the objective function. For example, on valley-shaped (e.g., Three-Hump Camel) and bowl-shaped (e.g., Bohachevsky) landscapes, configurations using quadratic mean functions generally performed better. This indicates that aligning the structure of the prior with the geometry of the target function can improve optimization outcomes.
4. **Robustness of Thompson Sampling**
   Thompson Sampling (TS) demonstrated remarkable robustness across all tested functions. It consistently performed well and did not encounter any failures or numerical issues. Its stochastic nature allows for effective exploration, making it a reliable choice for a wide range of optimization problems.

These findings underscore the importance of selecting appropriate acquisition functions based on the specific characteristics of the optimization problem at hand. While complex strategies may offer advantages in certain scenarios, simpler methods often provide sufficient performance with reduced computational overhead.

6 Conclusion

In this paper, we evaluated eight acquisition functions for single-objective Bayesian optimization under noisy conditions, using a comprehensive experimental design that combined multiple GP model configurations across five representative benchmark functions.

Our findings highlight that **no single acquisition function is universally optimal**. Instead, their performance varies significantly depending on the complexity of the objective function. For simple

convex landscapes, most acquisition functions—regardless of their sophistication—can reach near-optimal performance quickly. In contrast, for more challenging or deceptive functions, acquisition strategies that encourage exploration—such as Thompson Sampling and qLogNoisyExpectedImprovement—tend to perform more reliably.

Notably, Thompson Sampling was the only method that succeeded across all experiments without failure, offering a favorable trade-off between simplicity, robustness, and computational cost. In contrast, qKG and qPES were excluded from most comparisons due to excessive runtime, limiting their practical usability in time-sensitive or complex scenarios.

These results underscore the need to **match acquisition function choice to problem structure and resource constraints**. Simpler methods may suffice for low-dimensional or smooth problems, while more exploratory strategies are essential for noisy or multimodal landscapes.

Future work may extend this analysis to multi-objective settings, explore adaptive model selection, and improve the efficiency of entropy-based methods to enhance their applicability in realistic optimization tasks.

Appendix A: Summary of Optimization Execution Status Across Scenarios

This section presents a summary of the optimization execution outcomes. The tables below **selectively highlight configurations that encountered execution issues**, including timeouts and failures, during the Bayesian optimization process.

We classify unsuccessful runs into two categories:

- **Timeout**: This indicates that the optimization process was terminated prematurely due to excessive computation time. This situation occurred mainly for **qKG** and **qPES** on more complex objective functions (*Ackley*, *Easom*, and *Three-Hump Camel*), where the computational cost of acquisition function evaluation and optimization was prohibitively high.
- **Fail**: This refers to runtime failures that occurred either:
  1. **During Gaussian Process model updates**, often because the computed covariance matrix was not positive definite. Although BoTorch attempts to resolve such issues by **automatically adding jitter** (a small value added to the diagonal), repeated failure to restore numerical stability results in an exception.
  2. **During acquisition function optimization**, where optimization routines (e.g., L-BFGS-B or Adam) may crash due to malformed acquisition landscapes or numerical instabilities.

These failures are rare but highlight the sensitivity of some acquisition function and model combinations to numerical robustness. All failures and timeouts are explicitly labeled in the tables for transparency and future reproducibility.

Table 4: Execution Summary for Ackley Function (Success, Timeout, and Failures)

Acquisition Function	**Kernel**	Mean		
		Constant	Linear	Quadratic
qKG	**RBF**	timeout	timeout	timeout
	Matern(3/2)	timeout	timeout	timeout
	Matern(5/2)	timeout	timeout	timeout
qLogNoisyEI	**RBF**	complete	complete	fail[2]
	Matern(3/2)	complete	complete	complete
	Matern(5/2)	complete	complete	complete
qPES	**RBF**	timeout	timeout	timeout
	Matern(3/2)	timeout	timeout	timeout
	Matern(5/2)	timeout	timeout	timeout

Table 5: Execution Summary for Bohachevsky Function (Success, Timeout, and Failures)

Acquisition Function	Kernel	Mean		
qPES	**RBF**	complete	fail[1]	complete
	Matern(3/2)	complete	complete	complete
	Matern(5/2)	complete	fail[1]	complete

Table 6: Execution Summary for Booth Function (Success, Timeout, and Failures)

Acquisition Function	Kernel	Mean		
		Constant	Linear	Quadratic
qKG	**RBF**	complete	complete	timeout
	Matern(3/2)	complete	complete	timeout
	Matern(5/2)	complete	complete	timeout
qPES	**RBF**	complete	timeout	timeout
	Matern(3/2)	complete	complete	timeout
	Matern(5/2)	complete	timeout	timeout

Appendix B: Impact of Candidate Set Size on Optimization Stability and Performance

In this section, we examine the effect of varying the **candidate set size**—that is, the number of discretized evaluation points used when computing acquisition function values during Bayesian optimization. This parameter, denoted in our code as `num_candidates`, corresponds to the **batch size of sampled input points** used for **Monte Carlo approximation or maximization** of the acquisition function over the domain. It plays a critical role in acquisition strategies that rely on candidate-based search, especially those using sampling methods or discrete maximization rather than closed-form solutions.

We investigated this effect using three acquisition functions: **qLowerBoundMaxValueEntropy**, **qMaxValueEntropy**, and **Thompson Sampling**. For each method, we conducted experiments with three different candidate set sizes: **500**, **1000**, and **5000**. As shown in Figures 7, 8, and 9, the size of the candidate set directly influenced the stability and quality of optimization results.

Our findings indicate that:

- When using **only 500 candidates**, the optimization outcomes were noticeably unstable, with higher variance across replications and slower convergence.

Table 7: Execution Summary for Three-Hump Camel Function (Success, Timeout, and Failures)

Acquisition Function	Kernel	Mean		
		Constant	Linear	Quadratic
qKG	**RBF**	timeout	timeout	timeout
	Matern(3/2)	timeout	timeout	timeout
	Matern(5/2)	timeout	timeout	timeout
qLogNoisyEI	**RBF**	fail[2]	complete	complete
	Matern(3/2)	complete	complete	complete
	Matern(5/2)	complete	complete	complete
qPES	**RBF**	timeout	timeout	timeout
	Matern(3/2)	timeout	timeout	timeout
	Matern(5/2)	timeout	timeout	timeout

Table 8: Execution Summary for Easom Function (Success, Timeout, and Failures)

Acquisition Function	Kernel	Mean		
		Constant	Linear	Quadratic
qKG	**RBF**	timeout	timeout	timeout
	Matern(3/2)	timeout	timeout	timeout
	Matern(5/2)	timeout	timeout	timeout
qLBMVE	**RBF**	complete	fail[1]	fail[1]
	Matern(3/2)	complete	fail[1]	fail[1]
	Matern(5/2)	complete	fail[1]	fail[1]
qLogNoisyEI	**RBF**	complete	fail[1]	complete
	Matern(3/2)	complete	fail[1]	fail[1]
	Matern(5/2)	complete	fail[1]	fail[1]
qPES	**RBF**	timeout	timeout	timeout
	Matern(3/2)	timeout	timeout	timeout
	Matern(5/2)	timeout	timeout	timeout
qSR	**RBF**	fail[2]	fail[2]	fail[2]
	Matern(3/2)	fail[2]	fail[2]	fail[2]
	Matern(5/2)	complete	fail[2]	fail[2]
qUCB	**RBF**	complete	fail[2]	fail[2]
	Matern(3/2)	complete	fail[1]	fail[1]
	Matern(5/2)	complete	fail[1]	fail[1]

- Increasing the size to **1000 candidates** improved the performance significantly, yielding smoother trajectories and reduced variability.
- The **best results were observed with 5000 candidates**, where both the mean performance and variance were consistently optimal across the three acquisition functions.

These results demonstrate that candidate set resolution is essential for ensuring reliable optimization, particularly in acquisition functions that depend on discrete candidate evaluation, such as entropy-based and sampling-based methods. Based on our experiments, we recommend using at least **1000 candidate points** in practice, especially when working with noisy or multimodal objective functions.

Appendix C: Effect of Numerical Precision on Optimization Performance

We further investigated how the choice of numerical precision affects optimization outcomes, particularly in the context of Gaussian Process modeling and acquisition function evaluation. In most libraries, including BoTorch and GPyTorch, computations can be performed using either `float32` (single precision) or `float64` (double precision).

Figures 10 and 11 show the performance comparison of Bayesian optimization under two precision settings: `float32` and `float64`. These tests were conducted using Thompson Sampling on the **Ackley** and **Easom** functions, with fixed kernel and mean configurations.

The results reveal that:

- For the **Ackley function** (Figure 10), the difference between `float32` and `float64` was minimal, and both configurations converged to similar objective values with comparable variance. This suggests that for some moderately complex functions, single precision may suffice.
- In contrast, for the **Easom function** (Figure 11), `float64` clearly outperformed `float32`, achieving faster and more stable convergence. This function is characterized by a sharp, narrow global optimum, making it more sensitive to numerical precision.

These findings suggest that the **impact of precision is problem-dependent**. While `float32` may be adequate for smooth or well-behaved functions, more numerically challenging functions may benefit from higher precision.

We recommend using `float64` when computational resources permit, as it not only improves optimization accuracy in complex landscapes but also helps reduce the likelihood of **numerical instabilities** (e.g., non-positive definite covariance matrices or optimization failures during acquisition maximization), even though it may not eliminate such issues entirely.

Appendix D: Benchmark Functions

The five benchmark functions selected for this study are as follows:

- **Ackley 2:** Characterized by many local minima, representing highly multi-modal landscapes. The mathematical formula is:

$$f(x_1, x_2) = -20 \exp\left(-\frac{1}{5}\sqrt{\frac{1}{2}(x_1^2 + x_2^2)}\right) - \exp\left(\frac{1}{2}(\cos(2\pi x_1) + \cos(2\pi x_2))\right) + 20 + \exp(1) \tag{6}$$

- **Bohachevsky :** A bowl-shaped function with a single global minimum and smooth curvature. The mathematical formula is:

$$f(x_1, x_2) = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1) - 0.4\cos(4\pi x_2) + 0.7 \tag{7}$$

- **Booth:** A plate-shaped function with shallow gradients and a simple global minimum. The mathematical formula is:

$$f(x_1, x_2) = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2 \tag{8}$$

- **Easom :** Contains steep ridges and sharp drops, making it difficult for local models to capture global trends. The mathematical formula is:

$$f(x_1, x_2) = -\cos(x_1)\cos(x_2)\exp\left(-(x_1 - \pi)^2 - (x_2 - \pi)^2\right) \tag{9}$$

- **Three-Hump Camel:** A valley-shaped function with multiple inflection points and moderate complexity. The mathematical formula is:

$$f(x_1, x_2) = 2x_1^2 - 1.05x_1^4 + \frac{x^6}{6} + x_1x_2 + x_2^2 \tag{10}$$

These benchmark functions were chosen to represent a diverse range of optimization landscapes, including multi-modal, bowl-shaped, plate-shaped,rugged, and valley-shaped landscapes. Each function has a unique mathematical form and poses different challenges for optimization algorithms. By evaluating the performance of the acquisition functions on these benchmark functions, we can gain insights into their strengths and weaknesses in different optimization scenarios.

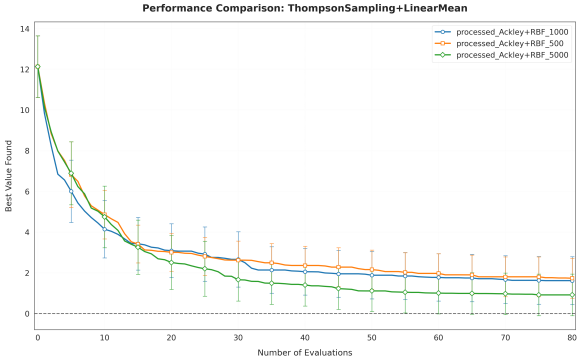


Figure 7: Effect of Candidate Set Size on Thompson Sampling Performance (Linear Mean + RBF Kernel)

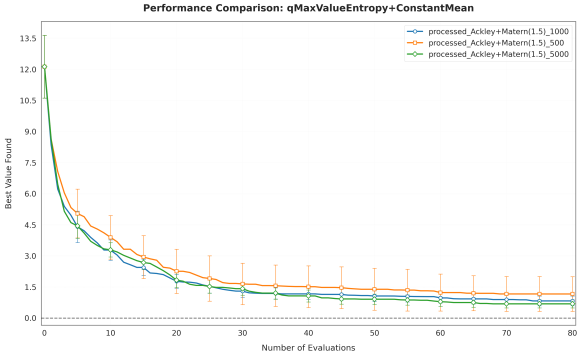


Figure 8: Effect of Candidate Set Size on qMaxValueEntropy Performance (Constant Mean + Matern(1.5) Kernel)

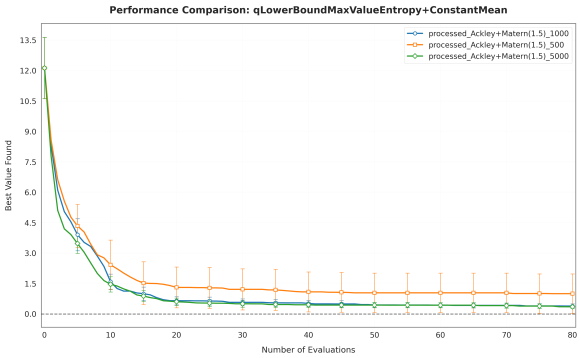


Figure 9: Effect of Candidate Set Size on qLowerBoundMaxValueEntropy Performance (Constant Mean +Matern(1.5) Kernel)

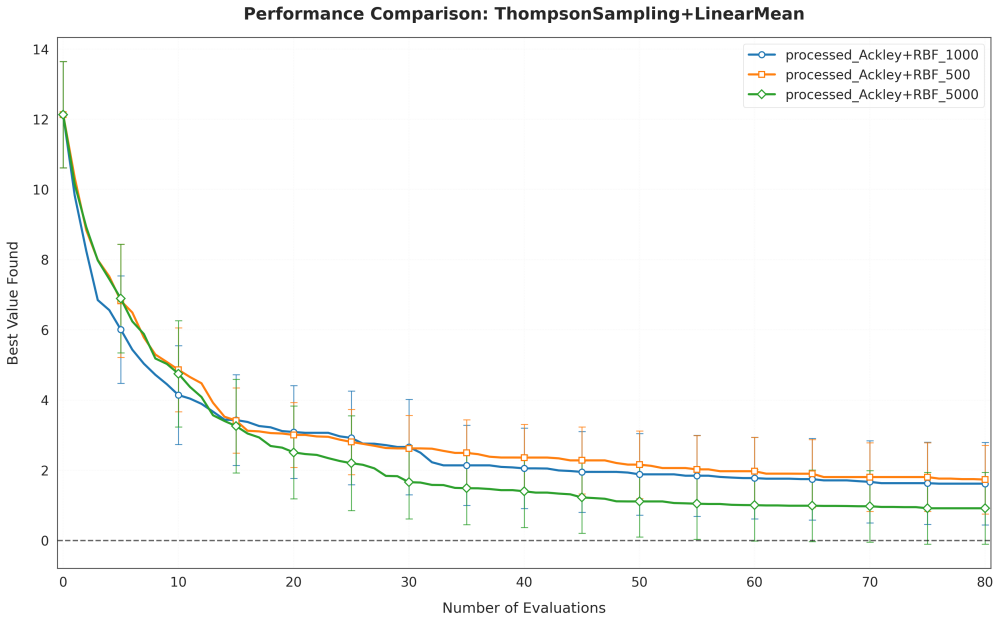


Figure 10: Performance Comparison on Ackley Function using Float32 vs. Float64 Precision(Thompson Sampling + Linear Mean + RBF Kernel)

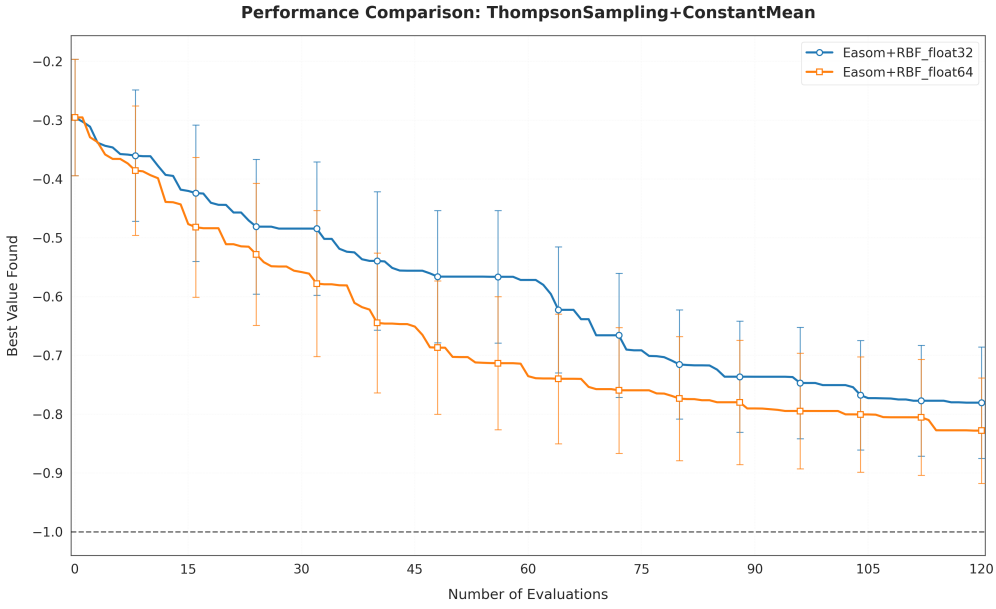


Figure 11: Performance Comparison on Easom Function using Float32 vs. Float64 Precision (Thompson Sampling + Constant Mean + RBF Kernel)

References

Maximilian Balandat, Brian Karrer, Daniel R. Jiang, Samuel Daulton, Benjamin Letham, Andrew Gordon Wilson, and Eytan Bakshy. Botorch: Programmable bayesian optimization in pytorch. *CoRR*, abs/1910.06403, 2019. URL http://arxiv.org/abs/1910.06403.

Maximilian Balandat, Brian Karrer, Daniel Jiang, Samuel Daulton, Ben Letham, Andrew G Wilson, and Eytan Bakshy. Botorch: A framework for efficient monte-carlo bayesian optimization. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 21524–21538. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/f5b1b89d98b7286673128a5fb112cb9a-Paper.pdf.

Peter I. Frazier, Warren B. Powell, and Savas Dayanik. A knowledge-gradient policy for sequential information collection. *SIAM Journal on Control and Optimization*, 47(5):2410–2439, 2008. doi: 10.1137/070693424. URL https://doi.org/10.1137/070693424.

Henry B. Moss, David S. Leslie, Javier Gonzalez, and Paul Rayson. GIBBON: general-purpose information-based bayesian optimisation. *CoRR*, abs/2102.03324, 2021. URL https://arxiv.org/abs/2102.03324.

Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando de Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1): 148–175, 2016. doi: 10.1109/JPROC.2015.2494218.

WILLIAM R THOMPSON. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3-4):285–294, 12 1933. ISSN 0006-3444. doi: 10.1093/biomet/25.3-4.285. URL https://doi.org/10.1093/biomet/25.3-4.285.

Zi Wang and Stefanie Jegelka. Max-value entropy search for efficient Bayesian optimization. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 3627–3635. PMLR, 06–11 Aug 2017. URL https://proceedings.mlr.press/v70/wang17e.html.

James T. Wilson, Riccardo Moriconi, Frank Hutter, and Marc Peter Deisenroth. The reparameterization trick for acquisition functions, 2017. URL https://arxiv.org/abs/1712.00424.

Jian Wu and Peter Frazier. The parallel knowledge gradient method for batch bayesian optimization. *Advances in neural information processing systems*, 29, 2016.

Table A1: Performance of Noisy Bayesian Optimization with Different Acquisition Functions on Test Functions

				Ackley (optimal value =0.0)				Bohachevsky (optimal value =0.0)				Booth (optimal value =0.0)				Easom(optimal value =-1.0)				Three Hump Camel (optimal value =0.0)			
				25%	50%	75%	100%	25%	50%	75%	100%	25%	50%	75%	100%	25%	50%	75%	100%	25%	50%	75%	100%
qSR	Constant	RBF	mean	5.26837386	4.55083874	4.24504532	4.23794638	0.07388865	0.07203123	0.07118676	0.071027684	0.00270056	0.00269863	0.00269129	0.00269129	\	\	\	\	0.39465637	0.22835229	0.19831407	0.16429281
			var	4.31995981	4.62800259	4.74242971	4.74572882	0.06227427	0.06269046	0.06287388	0.06294359	0.00230954	0.0023088	0.00230641	0.00230641	\	\	\	\	0.25917843	0.1814818	0.16878313	0.15344598
		Matern(3/2)	mean	7.61776305	7.19800825	6.99306251	6.91559104	2.03658638	1.94775669	1.85648393	1.763975661	0.31391823	0.28179224	0.14167607	0.10938267	\	\	\	\	0.34463317	0.28660712	0.27899417	0.27796217
			var	4.41464513	4.64313884	4.71469436	4.74349699	1.55057098	1.54960676	1.5621338	1.535803657	0.6150475	0.56810794	0.21735845	0.20565004	\	\	\	\	0.23371948	0.21445052	0.20806039	0.20728119
		Matern(5/2)	mean	6.12988077	5.58756491	5.3846497	5.23024046	0.11409084	0.06786771	0.05809968	0.057247572	0.0072558	0.00536528	0.00504897	0.00500909	-0.3810611	-0.4493642	-0.4540077	-0.4557483	0.37712446	0.2554956	0.25157343	0.23879742
			var	4.23729457	4.42516976	4.48045236	4.51724771	0.15759922	0.0921119	0.07565674	0.075110888	0.00735193	0.00554453	0.00534768	0.005363	0.31388032	0.36312361	0.36226766	0.36026979	0.24979199	0.22528316	0.22772362	0.2297595
	Linear	RBF	mean	5.26496885	5.05634512	4.90423493	4.69366689	0.67917127	0.65541606	0.5858265	0.562732902	0.04509324	0.03864458	0.03567823	0.03021789	\	\	\	\	0.40229399	0.24301422	0.22617636	0.1909988
			var	4.53653134	4.66425375	4.68602846	4.74830667	0.52663757	0.50739074	0.47174347	0.446711781	0.06868121	0.05828421	0.05319508	0.04207656	\	\	\	\	0.25251805	0.19972384	0.18426204	0.15525908
		Matern(3/2)	mean	8.22954031	7.80448164	7.61740267	7.51874116	2.07977096	1.99427627	1.91617804	1.815298158	0.40062771	0.3427614	0.1860522	0.14623584	\	\	\	\	0.364647	0.2551663	0.24535162	0.24226041
			var	4.80095139	4.93792299	5.12634931	4.99908144	1.52779237	1.53285907	1.55590162	1.527203492	0.73351176	0.61502147	0.31911593	0.30742743	\	\	\	\	0.22999223	0.19983176	0.18941752	0.19079521
		Matern(5/2)	mean	4.02199802	5.16358817	4.71917438	4.58390094	0.19076978	0.14794168	0.12699676	0.122860386	0.01291444	0.00860045	0.00660499	0.0065478	\	\	\	\	0.37113734	0.26150819	0.21874012	0.20905414
			var	4.77110811	4.09864117	4.29799815	4.3504827	0.2210902	0.19910657	0.17941123	0.170722809	0.02152427	0.0142359	0.00935287	0.00937687	\	\	\	\	0.22650497	0.21676745	0.19634652	0.18525515
	Quadratic	RBF	mean	5.54073305	5.04937433	4.85983239	4.2843532	3.44E-05	3.38E-05	3.31E-05	3.29E-05	0.11325363	0.09753397	0.08930984	0.08802056	\	\	\	\	0.34251625	0.18809477	0.18023277	0.17861555
			var	3.45048611	3.35599878	3.4608175	3.08941265	3.89E-05	3.88E-05	3.85E-05	3.85E-05	0.11242391	0.09158122	0.0848524	0.08376049	\	\	\	\	0.27331048	0.21843318	0.22066029	0.22171462
		Matern(3/2)	mean	7.18677431	6.94078396	6.62427541	6.45474484	3.25E-05	3.17E-05	3.11E-05	3.08E-05	0.3296142	0.26186412	0.20601766	0.17079664	\	\	\	\	0.26049043	0.11231161	0.10930283	0.10669316
			var	3.23677943	3.33314958	3.64469394	3.75076923	3.62E-05	3.58E-05	3.55E-05	3.55E-05	0.4563743	0.42908469	0.37182149	0.31840123	\	\	\	\	0.19785885	0.15155196	0.15041857	0.15046302
		Matern(5/2)	mean	7.13720049	6.59342971	6.19481573	5.77375812	3.42E-05	3.34E-05	3.29E-05	3.26E-05	0.01821512	0.012209	0.01009002	0.00996584	\	\	\	\	0.30740748	0.16377675	0.12903475	0.10605489
			var	2.92450388	2.91680498	2.77733057	3.13112929	3.89E-05	3.87E-05	3.85E-05	3.85E-05	0.02227419	0.01965029	0.01840506	0.01834404	\	\	\	\	0.22211489	0.20015449	0.14759756	0.14063753
qUCB	Constant	RBF	mean	3.69429923	2.73204959	1.99803053	1.78633566	0.05625901	0.05189356	0.04942978	0.048653296	0.0021678	0.00214411	0.00214059	0.00213988	-0.4041812	-0.4797065	-0.5387429	-0.5913407	0.41346759	0.15911521	0.11064719	0.07663301
			var	3.8646208	2.97927243	1.64996624	1.59797493	0.04876767	0.04521731	0.04419973	0.044028005	0.00185556	0.00186594	0.00186858	0.00186932	0.34916307	0.32427426	0.31711164	0.32804928	0.25279216	0.15419037	0.12556463	0.10406031
		Matern(3/2)	mean	4.8290646	4.03424096	3.59629274	3.38087498	2.00617459	1.7089645	1.16333498	1.092470702	0.25726494	0.06371844	0.04115324	0.02400818	-0.4024467	-0.5001985	-0.5576041	-0.6060872	0.28080057	0.11040547	0.041388	0.01913197
			var	4.02963118	4.22757569	4.04010858	4.08909353	1.76529292	1.10330309	0.49353457	0.439924829	0.28330042	0.06747375	0.0535763	0.02012166	0.31065486	0.28624532	0.27854788	0.26909758	0.18160733	0.07327746	0.03796591	0.02259397
		Matern(5/2)	mean	3.99678055	2.60404578	2.23954828	2.11755582	0.21527065	0.13811385	0.10730083	0.099882968	0.00836177	0.00533602	0.00513424	0.0050025	-0.4124415	-0.4387298	-0.4978437	-0.5240845	0.32231804	0.05862621	0.01842321	0.01356631
			var	4.05143293	3.45238781	3.48639222	3.5191206	0.21725476	0.14341335	0.10953429	0.102483438	0.01172101	0.00695137	0.00676378	0.00659046	0.31068326	0.32406121	0.33728462	0.33456339	0.22194465	0.05508884	0.0217861	0.01415446
	Linear	RBF	mean	4.02168532	3.19483331	2.94966981	2.84527918	0.65381982	0.62641385	0.55468588	0.530978563	0.04402278	0.03837307	0.03504233	0.03196384	\	\	\	\	0.41124054	0.15237622	0.13535745	0.11076741
			var	4.47007998	3.93142889	3.98544819	4.01505356	0.54336325	0.52458478	0.48099898	0.454802647	0.06748525	0.05851269	0.05338818	0.04629756	\	\	\	\	0.23978825	0.15124692	0.14994979	0.13690458
		Matern(3/2)	mean	5.88998606	5.04809021	4.74124727	4.29268635	2.07399396	1.76698526	1.1675894	1.11470613	0.32052031	0.078426	0.04863124	0.02603943	\	\	\	\	0.3048278	0.09823593	0.03372444	0.01697456
			var	4.68844344	4.81987891	4.7759495	4.97158848	1.77435032	1.14588936	0.54509028	0.490341326	0.33614068	0.07896841	0.06182191	0.02179025	\	\	\	\	0.20382845	0.07057911	0.03002188	0.01379191
		Matern(5/2)	mean	3.39828672	2.23173167	1.90550782	1.64858008	0.29378415	0.20844552	0.17573556	0.168021163	0.01194268	0.00693298	0.00514341	0.00499214	\	\	\	\	0.32811637	0.04349094	0.02139075	0.01276882
			var	3.35188935	2.63345488	2.62984531	2.69362094	0.25489963	0.20555166	0.19244843	0.188712597	0.0172445	0.00980115	0.00712704	0.00697238	\	\	\	\	0.21631121	0.0449685	0.02178357	0.01417401
	Quadratic	RBF	mean	5.05819268	4.34775842	3.90049905	3.85203549	3.98E-05	3.85E-05	3.71E-05	3.65E-05	0.11312772	0.09777327	0.08779381	0.08571523	\	\	\	\	0.34205704	0.110433	0.0915434	0.08012546
			var	3.65255077	3.13813098	3.06614463	3.08210438	4.74E-05	4.65E-05	4.48E-05	4.42E-05	0.11365723	0.092999	0.08189121	0.07914422	\	\	\	\	0.23978028	0.16022956	0.1618416	0.15599081
		Matern(3/2)	mean	6.67698369	6.5377914	6.37601483	6.04489179	3.6811E-05	3.6124E-05	3.5142E-05	3.47264E-05	0.15865703	0.04700093	0.02948017	0.01980897	\	\	\	\	0.30449219	0.1368113	0.05745209	0.05311476
			var	3.57817546	3.54777876	3.44519715	3.12114697	4.1795E-05	4.1487E-05	4.0462E-05	4.02295E-05	0.16579796	0.05355718	0.03511203	0.01787402	\	\	\	\	0.22534056	0.15013982	0.11389825	0.1148366
		Matern(5/2)	mean	6.48733781	6.00292052	5.80792234	5.58595228	3.5065E-05	3.4223E-05	3.3205E-05	3.28086E-05	0.02200135	0.01293284	0.0102202	0.01006089	\	\	\	\	0.25624866	0.11274574	0.06553808	0.04380227
			var	3.74069655	3.98586154	4.1625304	4.10205434	4.1525E-05	4.1002E-05	3.9802E-05	3.97786E-05	0.02843643	0.02322993	0.02190479	0.02187199	\	\	\	\	0.20687157	0.18675507	0.15602962	0.11607606
qLogNoisyEI	Constant	RBF	mean	0.84927509	0.28679362	0.14097383	0.0853776	0.07995945	0.07958022	0.06867688	0.062119681	0.00275854	0.00275482	0.0026214	0.00247225	-0.3993134	-0.4542471	-0.5605359	-0.5852545	\	\	\	\
			var	1.56600043	0.51956782	0.12536034	0.05958778	0.06340359	0.06372357	0.0581801	0.051116388	0.00231058	0.00230083	0.00213034	0.00183986	0.35226209	0.36495561	0.34857758	0.34512282	\	\	\	\
		Matern(3/2)	mean	1.28501895	0.65820526	0.61424463	0.58259413	24.1629376	16.5722116	7.79995376	5.050894713	1.10963382	0.92408051	0.7605355	0.48811684	-0.4273041	-0.5056548	-0.5873395	-0.6406611	0.29474222	0.20754542	0.12916408	0.08193246
			var	3.69435431	3.16683174	3.10684349	3.04365583	21.4256035	13.8162152	6.79904541	3.712849259	0.84286703	0.73677033	0.55732406	0.37095032	0.36986676	0.36164974	0.32955728	0.29858965	0.17747867	0.108809	0.10317989	0.05996981
		Matern(5/2)	mean	1.06336834	0.75199129	0.65562753	0.62455525	0.34897114	0.30225515	0.19686468	0.176984143	0.02332856	0.01551528	0.01527505	0.01527505	-0.3991387	-0.4879315	-0.5889491	-0.635688	0.31453724	0.17906183	0.14262288	0.08209317
			var	3.33216364	3.22139618	3.09626783	3.10193609	0.37872386	0.32698237	0.23969579	0.23294371	0.03372692	0.01170202	0.0115718	0.0115718	0.35176569	0.33905721	0.33412969	0.3217138	0.19273813	0.10569795	0.0888635	0.06796748
	Linear	RBF	mean	1.22930215	0.79920136	0.74139085	0.70006008	0.62021176	0.48470517	0.38348385	0.316962632	0.05356897	0.04038261	0.02261457	0.01755562	\	\	\	\	0.39587773	0.26101149	0.13050376	0.0684184
			var	3.07735814	3.09392236	3.10440438	3.11139153	0.47100609	0.40323082	0.32367439	0.227253828	0.09144976	0.06449225	0.03191264	0.02242293	\	\	\	\	0.26737144	0.19089044	0.18558583	0.13642218
		Matern(3/2)	mean	1.23376879	0.71363884	0.67392535	0.65707265	24.4516897	16.7991741	7.80527072	5.210487666	1.14933988	0.90547143	0.7800889	0.58463162	\	\	\	\	0.29244964	0.22044703	0.15413086	0.09512077
			var	3.72910703	3.45806995	3.46537863	3.46852947	21.5988365	13.9396232	6.30200961	3.663789873	0.81210817	0.76335707	0.53565354	0.39246072	\	\	\	\	0.18907572	0.1289844	0.10660933	0.07449734
		Matern(5/2)	mean	1.05695241	0.75826558	0.71271859	0.67857103	0.43310479	0.42201486	0.37985887	0.362758584	0.03549003	0.02203216	0.02157275	0.01893816	\	\	\	\	0.340082	0.16756915	0.12516328	0.08403459
			var	3.34027684	3.36975554	3.37785668	3.38380506	0.42682478	0.40107301	0.37459884	0.374300026	0.06575344	0.02362646	0.0231347	0.0172653	\	\	\	\	0.22292184	0.1081426	0.08635539	0.05076707
	Quadratic	RBF	mean	\	\	\	\	3.57E-05	3.55E-05	3.52E-05	3.49E-05	0.0404457	0.02272238	0.01606328	0.01162275	-0.350928	-0.3586835	-0.3636088	-0.3843916	0.31909952	0.13031418	0.07282889	0.06312951
			var	\	\	\	\	3.91E-05	3.92E-05	3.93E-05	3.94E-05	0.03450282	0.01635437	0.01082781	0.00843307	0.3161216	0.32902024	0.33103932	0.33861818	0.20335909	0.15985238	0.1583221	0.15478565
		Matern(3/2)	mean	1.81563409	0.55037298	0.46559929	0.34826265	2.56E-05	2.17E-05	2.09E-05	2.08E-05	0.68974994	0.49402249	0.46139855	0.35484053	\	\	\	\	0.27164447	0.17515799	0.13432974	0.09419599
			var	2.25633852	1.46443537	1.42337446	1.01011038	2.87E-05	2.55E-05	2.55E-05	2.55E-05	0.56109771	0.37134653	0.32653775	0.26415858	\	\	\	\	0.20901667	0.14536592	0.1233017	0.11539688
		Matern(5/2)	mean	2.22874985	0.74358758	0.44533853	0.41032177	3.29E-05	3.27E-05	3.25E-05	3.23E-05	0.08371519	0.03512752	0.03256631	0.03256631	\	\	\	\	0.27632268	0.18005838	0.11592282	0.06697508
			var	3.07266558	1.78156723	1.52756236	1.52092046	3.57E-05	3.58E-05	3.58E-05	3.59E-05	0.09968198	0.02696591	0.02560474	0.02560474	\	\	\	\	0.22817658	0.16212449	0.15392458	0.1146766
qPredictiveEntropySearch	Constant	RBF	mean	\	\	\	\	581.944009	472.796583	285.003803	121.1270216	0.82682157	0.28122916	0.16407982	0.1379513	\	\	\	\	\	\	\	\
			var	\	\	\	\	510.679063	524.330591	491.19176	221.1745379	1.0190278	0.32423642	0.18106797	0.13325864	\	\	\	\	\	\	\	\
		Matern(3/2)	mean	\	\	\	\	581.944009	581.944009	506.79866	454.8607976	16.0051293	12.6922419	7.29463653	4.80789151	\	\	\	\	\	\	\	\
			var	\	\	\	\	510.679063	510.679063	410.570434	374.1165836	11.3899029	8.04735055	4.97998083	2.95991294	\	\	\	\	\	\	\	\
		Matern(5/2)	mean	\	\	\	\	581.944009	581.944009	547.962473	396.5604131	1.46643315	0.85400555	0.43116286	0.30414639	\	\	\	\	\	\	\	\
			var	\	\	\	\	510.679063	510.679063	524.275891	399.2710449	1.84218722	0.78419553	0.39145266	0.23645934	\	\	\	\	\	\	\	\
	Linear	RBF	mean	\	\	\	\	\	\	\	\	0.29952184	0.15721926	0.10409227	0.05775134	\	\	\	\	\	\	\	\
			var	\	\	\	\	\	\	\	\	0.49809071	0.26740868	0.16304757	0.06135956	\	\	\	\	\	\	\	\
		Matern(3/2)	mean	\	\	\	\	581.944009	581.944009	506.792968	455.277953	16.0051293	13.2413285	7.78261574	5.09047281	\	\	\	\	\	\	\	\
			var	\	\	\	\	510.679063	510.679063	410.577554	374.4944243	11.3899029	8.38791914	5.70070736	3.92318236	\	\	\	\	\	\	\	\
		Matern(5/2)	mean	\	\	\	\	\	\	\	\	1.75790318	0.93400668	0.41912997	0.25617452	\	\	\	\	\	\	\	\
			var	\	\	\	\	\	\	\	\	1.69771329	0.85411522	0.4546005	0.24050739	\	\	\	\	\	\	\	\
	Quadratic	RBF	mean	\	\	\	\	0.05929026	0.00123403	0.00110135	0.000779266	0.12101713	0.09545368	0.08316857	0.07316169	\	\	\	\	\	\	\	\
			var	\	\	\	\	0.23662938	0.00376141	0.00353656	0.003250033	0.13044144	0.11256632	0.10210147	0.09294423	\	\	\	\	\	\	\	\
		Matern(3/2)	mean	\	\	\	\	0.62968145	0.62967449	0.62966506	0.629662987	15.9830472	9.84214375	4.58541295	3.32167643	\	\	\	\	\	\	\	\
			var	\	\	\	\	3.44872197	3.44872328	3.44872506	3.448725457	11.3793603	7.3554857	3.14003673	1.93915358	\	\	\	\	\	\	\	\
		Matern(5/2)	mean	\	\	\	\	35.278364	3.3850415	3.3195919	3.274422139	3.2484122	0.69616424	0.1783485	0.08896009	\	\	\	\	\	\	\	\
			var	\	\	\	\	191.714954	17.2857328	17.2867989	17.2902473	4.91021417	0.67991081	0.22841947	0.11414113	\	\	\	\	\	\	\	\
qMaxValueEntropy	Constant	RBF	mean	3.34601654	2.1514158	1.71837793	1.49269113	199.61496	80.551976	18.1928209	15.94480682	0.04734014	0.00768502	0.00487877	0.00386867	-0.3994554	-0.5088482	-0.5560263	-0.6170922	0.52848171	0.39553819	0.2159553	0.10813695
			var	2.30274152	1.07055437	0.91637114	0.87748173	339.771453	183.637636	36.8028609	35.74334537	0.19504396	0.00825516	0.00627677	0.00595118	0.3476966	0.35245261	0.33784225	0.31167151	0.365072	0.27175705	0.22039189	0.17242888
		Matern(3/2)	mean	1.75643819	0.96045644	0.9029612	0.86655003	128.839068	88.4930868	24.5890616	15.43579941	4.197713	2.62435733	1.57100793	1.43244151	-0.4944333	-0.5567771	-0.6268161	-0.6490087	0.49260758	0.40223978	0.34078058	0.29937342
			var	1.1539312	0.63105782	0.64063858	0.60813168	184.00495	173.837537	24.8845731	15.54840242	4.46332895	1.89534735	1.19189978	1.07023941	0.38272291	0.36749433	0.355925	0.35037112	0.31319847	0.26794448	0.21708217	0.20201266
		Matern(5/2)	mean	2.35435959	1.65106056	1.43453185	1.3001702	102.119496	44.7906734	1.15950724	1.045469169	0.08681395	0.05818324	0.04540594	0.04172581	-0.4799569	-0.5344962	-0.5971408	-0.6398151	0.4861308	0.27298153	0.231018	0.20363963
			var	1.17167718	1.02071847	0.87112393	0.85668436	269.442033	179.604875	0.98746366	0.801376045	0.10133551	0.09866463	0.08002761	0.07737737	0.35328226	0.35579563	0.35369629	0.33079393	0.32866589	0.23286026	0.1925885	0.16372117
	Linear	RBF	mean	3.93070592	2.86729045	2.67955856	2.49852003	129.383668	68.2267293	51.3129731	37.6839852	0.0472511	0.03824351	0.03464288	0.02550869	\	\	\	\	0.47644645	0.30751682	0.22017638	0.11230606
			var	3.6530788	2.79346287	2.81700412	2.87372166	242.797873	121.180937	115.349472	114.0226263	0.06688748	0.0549447	0.05217647	0.03551687	\	\	\	\	0.3239118	0.28351176	0.27177029	0.19203124
		Matern(3/2)	mean	2.42567729	1.17553856	0.8305675	0.73275839	143.031784	68.128831	22.1613858	14.81520023	5.81509144	2.58166581	1.73427317	1.36507694	\	\	\	\	0.47828993	0.35408574	0.304786	0.26029106
			var	1.41488391	0.89282685	0.62676319	0.55299447	264.700324	100.784313	25.9403391	17.35262182	6.94582943	1.63510812	1.06070751	0.96319493	\	\	\	\	0.31873814	0.22545051	0.17967259	0.15993651
		Matern(5/2)	mean	2.40939528	1.424841	1.10719373	1.03520052	111.904402	21.5559437	20.9126109	20.78357282	0.08887761	0.04306707	0.03690127	0.03570247	\	\	\	\	0.52079449	0.32862275	0.25884372	0.21811752
			var	1.35439799	0.87303751	0.89726668	0.77788893	272.550936	107.143914	107.24891	107.2729295	0.11141089	0.03643554	0.03735009	0.03556197	\	\	\	\	0.44247253	0.23886672	0.18333947	0.15253367
	Quadratic	RBF	mean	4.44068914	3.28491867	2.56738989	2.3384346	429.276886	388.357287	346.775992	317.7670146	0.16656597	0.13663608	0.10323489	0.09183373	\	\	\	\	0.23222086	0.17678475	0.06590274	0.05275861
			var	2.78324144	1.88675308	1.74229522	1.84076519	383.321658	383.370556	362.517407	334.4028407	0.19121543	0.16162951	0.11734965	0.10490747	\	\	\	\	0.21310321	0.17486632	0.06944816	0.04765317
		Matern(3/2)	mean	3.74373482	1.83591495	1.42024135	1.15348474	506.406428	410.710661	355.766119	339.3056841	2.53616061	1.72588706	1.11479359	0.84882974	\	\	\	\	0.28522327	0.21936561	0.16679015	0.14018462
			var	2.44678359	1.74101517	1.05860679	0.9268877	499.252196	358.492219	295.030665	299.6881718	2.86973223	2.14737578	0.84985416	0.71318599	\	\	\	\	0.27507353	0.24692085	0.19576866	0.14452774
		Matern(5/2)	mean	3.10997587	1.51483623	1.24279849	1.08927189	455.293269	417.871077	350.772809	350.7325993	0.17738369	0.10762536	0.06455276	0.05988488	\	\	\	\	0.22970491	0.22019057	0.14647389	0.13161529
			var	1.83894973	1.01524078	0.86239118	0.77890718	419.572212	431.408464	383.385002	383.4143015	0.25288357	0.14917162	0.10435981	0.1039447	\	\	\	\	0.25683665	0.23523264	0.16187271	0.14094904
qLowerBoundMaxValueEntropy	Constant	RBF	mean	2.54759879	2.07443938	1.78572934	1.51908525	14.7570402	2.67954996	0.27381772	0.240692275	0.00853075	0.00565154	0.00442631	0.00385063	-0.4277758	-0.5084264	-0.5437668	-0.6083474	0.47487617	0.34765843	0.14779217	0.06570919
			var	1.15784636	1.05818494	1.0307921	0.82195953	37.7308712	12.6804546	0.25022597	0.244213497	0.01248948	0.0061584	0.00538409	0.00535525	0.36615111	0.36459492	0.37403904	0.35505081	0.29736153	0.25282121	0.18036662	0.12912078
		Matern(3/2)	mean	0.73764007	0.54616978	0.47377777	0.46333766	66.6199035	35.0262911	15.2937325	8.101465594	3.48328265	2.27780303	1.58855498	1.22919213	-0.4543445	-0.5271138	-0.6087697	-0.6631829	0.39249221	0.24012458	0.21669524	0.15209441
			var	0.4647828	0.37891758	0.30474928	0.30457199	54.4578575	33.9707136	12.7002187	7.431274484	2.65491049	1.52245621	1.17695804	0.91990877	0.37882531	0.38154257	0.33914616	0.31323287	0.22768156	0.14719777	0.12775504	0.10828885
		Matern(5/2)	mean	1.38357816	0.78355281	0.73531132	0.68409874	34.4914793	9.36136439	5.12906866	5.025090304	0.07547894	0.05026755	0.0448072	0.04283451	-0.4339601	-0.483228	-0.5178005	-0.6116317	0.38323103	0.19290928	0.14467436	0.10718551
			var	1.06275512	0.60101721	0.56688426	0.56598449	94.5355396	44.6990132	22.6442376	22.66225982	0.09337883	0.09189753	0.0919042	0.09231142	0.37692544	0.36415275	0.36453568	0.33210943	0.25741333	0.12886176	0.10069965	0.07738392
	Linear	RBF	mean	3.08402095	2.2464669	1.80452594	1.59266768	28.8580906	9.13243152	3.26078335	0.634950406	0.30808441	0.03607571	0.03316116	0.02877259	\	\	\	\	0.45983349	0.35191172	0.17487021	0.09027762
			var	3.58225011	2.79753313	2.23247872	2.06133205	54.8521037	31.1542907	13.3052465	0.554964766	1.43409137	0.05036864	0.04776703	0.04315127	\	\	\	\	0.27358124	0.23414393	0.18669159	0.14976686
		Matern(3/2)	mean	0.80924694	0.67841973	0.49686443	0.44774647	66.1841542	41.1233182	17.5009099	8.178805302	3.47126815	2.36075771	1.68148198	1.2777606	\	\	\	\	0.36883022	0.21480288	0.19006942	0.166484
			var	0.63245985	0.59735173	0.39923632	0.38353248	54.4207282	39.9041807	14.1833661	7.336110654	2.45254351	1.54662066	1.10076164	0.88388386	\	\	\	\	0.24407722	0.14584203	0.11780119	0.11705723
		Matern(5/2)	mean	1.33416075	0.99144014	0.90216317	0.70806621	10.2906613	9.9493386	4.16496502	0.796048532	0.07060851	0.04394687	0.0360387	0.0351515	\	\	\	\	0.35771549	0.20984103	0.17740931	0.12525304
			var	0.85332477	0.74463634	0.68269965	0.54238545	32.5036	32.589439	17.1570948	0.542186746	0.07880418	0.04255072	0.03694547	0.0371407	\	\	\	\	0.22739668	0.1268138	0.10563076	0.08240066
	Quadratic	RBF	mean	3.15175762	2.54535635	1.94613328	1.76310091	348.936651	343.252747	293.836681	293.8364706	0.14417614	0.09193362	0.07656847	0.06419549	\	\	\	\	0.31190731	0.11468335	0.06495907	0.05317451
			var	2.5359176	2.01376911	1.62843592	1.46116138	421.680047	425.730241	405.977801	405.9779581	0.15951263	0.10719123	0.09905571	0.07879144	\	\	\	\	0.20957648	0.15613687	0.11694492	0.11472787
		Matern(3/2)	mean	1.73814727	1.54646319	1.50033336	1.31907461	426.207203	421.918384	371.533781	371.5337813	2.02850941	1.45643795	1.09248329	0.71540616	\	\	\	\	0.27744841	0.16578088	0.13745926	0.10778824
			var	2.79119605	2.79533145	2.72038047	2.403613	403.145834	407.077437	393.333916	393.3339158	1.34243706	1.18464811	0.84730733	0.5116864	\	\	\	\	0.23060514	0.14791131	0.13098282	0.12289839
		Matern(5/2)	mean	2.58198277	2.17097657	1.61588964	1.37466414	367.173884	359.560201	295.753834	295.7536192	0.12960051	0.09371704	0.07309954	0.04889951	\	\	\	\	0.30016034	0.18781513	0.14944116	0.11635798
			var	3.44114521	3.13688333	2.56162286	2.52445603	400.310061	406.470999	384.187292	384.187463	0.19634693	0.11977342	0.09826096	0.05700444	\	\	\	\	0.26434451	0.15828107	0.15319196	0.15167103
qKG	Constant	RBF	mean	\	\	\	\	570.342669	493.943861	484.594243	454.6337547	15.3453972	15.3453972	14.8854958	14.6713336	\	\	\	\	\	\	\	\
			var	\	\	\	\	499.127545	367.632147	359.320299	355.4578406	10.1723661	10.1723661	10.2397353	9.94094167	\	\	\	\	\	\	\	\
		Matern(3/2)	mean	\	\	\	\	155.068848	115.41622	45.4520146	35.41746604	2.00409207	1.6007643	0.94585788	0.54511734	\	\	\	\	\	\	\	\
			var	\	\	\	\	87.8565353	58.4557462	15.5428317	9.502380555	1.0558765	1.01685364	0.655603	0.29193861	\	\	\	\	\	\	\	\
		Matern(5/2)	mean	\	\	\	\	423.673149	391.16689	312.941303	279.9945902	6.9158471	5.53345729	4.63050782	4.24155424	\	\	\	\	\	\	\	\
			var	\	\	\	\	285.733599	251.149129	195.030532	177.3886655	2.86885104	1.98864186	1.56646751	1.43831132	\	\	\	\	\	\	\	\
	Linear	RBF	mean	\	\	\	\	581.944009	559.530283	423.654849	300.7011273	15.335327	14.3583446	14.2331141	13.4562272	\	\	\	\	\	\	\	\
			var	\	\	\	\	510.679063	520.062062	379.468635	365.7648998	10.5781865	8.98484896	9.07035073	8.37559826	\	\	\	\	\	\	\	\
		Matern(3/2)	mean	\	\	\	\	157.605512	118.084949	50.2399032	38.26577094	2.09348746	1.62974441	0.81373916	0.4506783	\	\	\	\	\	\	\	\
			var	\	\	\	\	91.9865061	57.4359526	16.8762284	11.91988807	1.04853646	1.04232602	0.44434939	0.21977997	\	\	\	\	\	\	\	\
		Matern(5/2)	mean	\	\	\	\	448.817917	418.66574	318.949816	281.184841	6.69940777	5.16158742	4.49209683	4.1718304	\	\	\	\	\	\	\	\
			var	\	\	\	\	306.470673	295.163767	256.753501	212.2646568	2.74404659	1.8659993	1.54938138	1.48103026	\	\	\	\	\	\	\	\
	Quadratic	RBF	mean	\	\	\	\	502.320016	435.375183	406.047792	379.1316877	15.1667434	15.1667434	13.2060461	11.9188655	\	\	\	\	\	\	\	\
			var	\	\	\	\	488.429256	358.271688	345.599744	321.7807044	10.0460311	10.0460311	8.34022646	7.16190801	\	\	\	\	\	\	\	\
		Matern(3/2)	mean	\	\	\	\	495.958505	409.064918	388.592036	361.6759321	1.68193336	1.38557473	0.76699667	0.52698853	\	\	\	\	\	\	\	\
			var	\	\	\	\	483.601999	352.048434	329.081995	302.3681757	0.88023148	0.71852755	0.35448289	0.34540297	\	\	\	\	\	\	\	\
		Matern(5/2)	mean	\	\	\	\	495.958505	408.203964	382.193842	355.2777382	6.19848981	4.74231273	3.90977203	3.79891629	\	\	\	\	\	\	\	\
			var	\	\	\	\	483.601999	351.625192	353.658919	328.4050538	2.62209136	1.76089043	1.37735339	1.34871473	\	\	\	\	\	\	\	\
ThompsonSampling	Constant	RBF	mean	2.80233124	1.6811761	1.27659025	0.76135671	1.7357755	1.39389682	1.01936418	0.95139459	0.04123242	0.020973	0.01283026	0.00983057	-0.5363122	-0.673681	-0.7658684	-0.8045193	0.45453139	0.28572431	0.11431438	0.06264467
			var	4.28535914	3.47117325	3.38274118	2.03379128	1.08393808	0.79288313	0.3894503	0.347704411	0.03901251	0.02042393	0.01235768	0.00793064	0.3294035	0.31349822	0.27536225	0.25593857	0.29237469	0.1743373	0.1145051	0.08830169
		Matern(3/2)	mean	1.54801804	0.60784558	0.37845396	0.30314113	70.3586449	28.9359083	18.160862	12.95937614	3.00587012	1.984472	1.48750287	0.91079422	-0.5016262	-0.6320621	-0.7555079	-0.7772092	0.36224843	0.25121947	0.19555847	0.16068309
			var	0.77626225	0.40325319	0.28782829	0.14883172	94.6891191	24.2427745	18.2870472	10.14030684	2.2980354	1.60582967	1.27367839	0.72909236	0.38200898	0.37432209	0.29898009	0.29634379	0.24223473	0.14246607	0.10841712	0.09420951
		Matern(5/2)	mean	1.31977561	0.47740015	0.32300446	0.27665182	3.11906036	1.92952797	1.42011858	1.251823766	0.2994727	0.09946338	0.07247098	0.05673887	-0.4787442	-0.587054	-0.6684413	-0.7253813	0.38771159	0.25446042	0.17159986	0.10975236
			var	0.89785603	0.32572054	0.21881261	0.16812124	2.3797292	0.9409679	0.64891603	0.570501112	0.44719552	0.08424027	0.06319826	0.04897387	0.35733594	0.35199852	0.31344893	0.30628915	0.25692594	0.16338936	0.12405887	0.07103196
	Linear	RBF	mean	2.46425407	1.34923734	0.94275023	0.8512482	1.8264454	1.42725342	1.01729633	0.970368673	0.08340615	0.04930369	0.02248063	0.0189182	-0.3729162	-0.4672348	-0.5508942	-0.6060942	0.4883571	0.28025792	0.14585391	0.06853643
			var	4.02267202	3.11636116	3.07423331	3.08580698	1.07853206	0.75734336	0.38572642	0.34515893	0.09772609	0.07905872	0.03082072	0.02509379	0.32481992	0.35335647	0.36094037	0.36362389	0.32125423	0.18022463	0.14943417	0.12124534
		Matern(3/2)	mean	2.19507009	0.64203848	0.40083931	0.31102257	74.3933282	30.8646447	18.160862	12.95937614	2.96694842	1.9455503	1.45505621	0.97467988	-0.4465765	-0.5649605	-0.7000446	-0.7358054	0.35310948	0.30612314	0.20567068	0.15811508
			var	2.84670743	0.41775598	0.2995037	0.15469494	95.7071465	26.9037592	18.2870472	10.14030684	2.33660807	1.63560652	1.29116726	0.85042547	0.34301169	0.35421334	0.33878443	0.3381085	0.22497868	0.19235559	0.12263329	0.12203334
		Matern(5/2)	mean	1.53145326	0.5451506	0.35101446	0.29042441	3.92545083	2.40301743	1.5658224	1.361833845	0.28206069	0.15917146	0.07308779	0.0595903	-0.4305261	-0.5416003	-0.6392953	-0.6594956	0.40561455	0.27367912	0.15918211	0.10356606
			var	1.38322126	0.34308368	0.20031979	0.15374932	3.03052533	1.34014757	0.84044836	0.75145127	0.32312366	0.2888742	0.06544664	0.04353942	0.3508684	0.38815668	0.36944698	0.37886748	0.25316175	0.16397515	0.12262101	0.08119699
	Quadratic	RBF	mean	2.78432753	1.90515871	1.24401869	1.06518547	1.68181923	1.3731489	1.02081153	0.901690077	0.10872309	0.07414018	0.03394377	0.02806619	-0.4088642	-0.4813629	-0.5288993	-0.5435498	0.32820088	0.13715176	0.0720455	0.05238349
			var	2.20524812	2.03254329	1.7444978	1.5757989	0.97401788	0.72664088	0.38902258	0.358133389	0.11169007	0.0910139	0.03527939	0.02435224	0.33551769	0.36958053	0.37822466	0.38777433	0.26181119	0.14049107	0.1290086	0.1191715
		Matern(3/2)	mean	2.69023542	1.49540163	1.00753486	0.71942141	1.68181923	1.3731489	1.01688154	0.897760092	2.89760272	1.79345752	1.32552434	0.84472733	-0.4244533	-0.5224053	-0.6446448	-0.6999583	0.30906335	0.21188449	0.1702531	0.12477507
			var	2.13893653	1.87182412	1.59271492	1.12458512	0.97401788	0.72664088	0.38928685	0.357066708	2.20611231	1.50669674	1.13287577	0.6564047	0.32662231	0.34492299	0.34365249	0.34389793	0.22058959	0.12495697	0.12147562	0.07927828
		Matern(5/2)	mean	2.64829539	1.3996353	0.81020203	0.66574987	1.68181923	1.3731489	1.01688154	0.897760092	0.37046805	0.14145252	0.08244953	0.06301391	-0.4051783	-0.5230367	-0.6103221	-0.6576805	0.33929119	0.2108887	0.1043301	0.07955419
			var	2.14931223	1.8930965	1.28187855	1.18488328	0.97401788	0.72664088	0.38928685	0.357066708	0.64530075	0.15024212	0.07569095	0.06441274	0.32594206	0.33811938	0.35335063	0.35623387	0.27050192	0.1574133	0.09709676	0.07795034

The table shows the mean and standard deviation (over 30 replications) of the best observed value at 25%, 50%, 75%, and 100% of the optimization budget (additional evaluations after initial sampling). Results are reported for 5 test functions with 8 acquisition functions, each combined with 3 mean and 3 covariance functions (9 variants per acquisition).