entropy

MDPI

Article

Advanced Monte Carlo for Acquisition Sampling in Bayesian Optimization

Javier Garcia-Barcos and Ruben Martinez-Cantin *

Instituto Universitario de Investigacion en Ingenieria de Aragon (I3A), Universidad de Zaragoza, 50018 Zaragoza, Spain; jgbarcos@unizar.es
* Correspondence: rmcantin@unizar.es

Abstract: Optimizing complex systems usually involves costly and time-consuming experiments, where selecting the experiments to perform is fundamental. Bayesian optimization (BO) has proved to be a suitable optimization method in these situations thanks to its sample efficiency and principled way of learning from previous data, but it typically requires that experiments are sequentially performed. Fully distributed BO addresses the need for efficient parallel and asynchronous active search, especially where traditional centralized BO faces limitations concerning privacy in federated learning and resource utilization in high-performance computing settings. Boltzmann sampling is an embarrassingly parallel method that enables fully distributed BO using Monte Carlo sampling. However, it also requires sampling from a continuous acquisition function, which can be challenging even for advanced Monte Carlo methods due to its highly multimodal nature, constrained search space, and possibly numerically unstable values. We introduce a simplified version of Boltzmann sampling, and we analyze multiple Markov chain Monte Carlo (MCMC) methods with a numerically improved log EI implementation for acquisition sampling. Our experiments suggest that by introducing gradient information during MCMC sampling, methods such as the MALA or CyclicalSGLD improve acquisition sampling efficiency. Interestingly, a mixture of proposals for the Metropolis–Hastings approach proves to be effective despite its simplicity.

Keywords: Bayesian optimization; Gaussian process; MCMC

Academic Editor: Eduardo C. Garrido-Merchán

Received: 27 November 2024
Revised: 3 January 2025
Accepted: 6 January 2025
Published: 10 January 2025

Citation: Garcia-Barcos, J.; Martinez-Cantin, R. Advanced Monte Carlo for Acquisition Sampling in Bayesian Optimization. *Entropy* **2025**, *27*, 58. https://doi.org/10.3390/e27010058

1. Introduction

Solving engineering and scientific problems often requires the use of complex computer models, setting up expensive prototypes, resource-intensive experimentation, or tuning the best parameters of a deep learning model. Identifying the best configuration for these processes requires evaluating various options with a trial-and-error approach. Given the high cost and time associated with these tasks, it is essential to achieve high sample efficiency. Although traditional optimization methods and evolutionary algorithms can be effective in finding the optimal configuration, they may not always be as sample-efficient as certain expensive contexts require. Carefully selecting which experiments to conduct is crucial. Bayesian optimization (BO) [1–3] is a method that helps streamline the process of selecting the most promising experiments and identifying optimal solutions as efficiently as possible.

BO builds a probabilistic surrogate model that approximates the unknown target function from observations. Then, an acquisition function based on the surrogate model is used to find which point should be observed next, balancing exploration and exploitation

to achieve sample efficiency. This process is intrinsically sequential, and the standard BO formulation is limited to single experiment evaluations.

Alternatively, many costly problems can be performed cheaper and more efficiently using a parallel BO strategy that enables parallel experimentation, such as chemical experiments in a wet lab for drug discovery [4], multi-agent experiments [5], and autoML in a cluster [6]. Parallel BO can be achieved with a batch setting [7,8], where multiple experiments can be proposed to be carried out in parallel, including asynchronous execution [9]. These BO parallel approaches are centralized, with a single central node typically carrying out the bulk of the BO computation of updating the surrogate model and proposing the batch of experiments, which will be evaluated by many worker nodes. This hinders maintaining privacy in federated learning [10] and resource utilization when experiments are carried out with high-performance computing [11]. To solve this, BO can be adapted to work in an asynchronous decentralized fashion [12,13]. This fully distributed workflow enables the experiment to be carried out even if there is a partial failure in the distributed network since data is shared asynchronously and new experiments can be proposed independently. Figure 1 compares them.

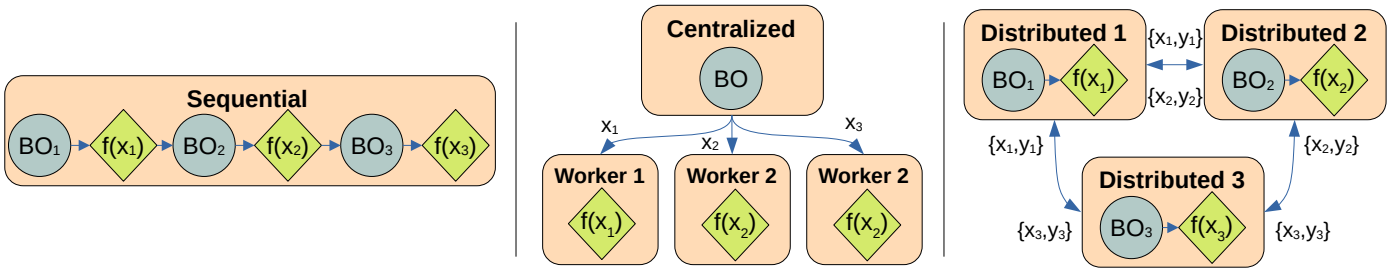

Figure 1. Types of parallelization schemes in Bayesian optimization (BO). The left side shows the sequential node, which is the standard BO approach, where BO computations and sample evaluation are strictly sequential and usually carried on a single node. The middle shows a diagram of parallel BO, with one node carrying the BO computations and multiple workers evaluating samples in parallel, including asynchronous evaluations. The right shows the fully distributed approach, where each node shares data asynchronously with other nodes while being capable of operating independently of each other. The lack of centralization and synchronization ensures the highest resource utilization possible.

A simple approach to achieving a fully distributed BO is replicating the BO algorithm across multiple nodes, sharing data asynchronously and, to prevent different nodes from suggesting the same exact experiments, relying on sampling approaches such as Thompson sampling [4] and Boltzmann sampling [13] with different random seeds. In particular, Boltzmann sampling has some interesting properties, such as being able to operate with existing well-understood and tested acquisition functions and offering better robustness to modeling inaccuracies [14]. Boltzmann sampling also requires setting a temperature parameter that depends on the acquisition function values. However, there is high variability in the acquisition values, since they can change considerably as the optimization progresses. This means that, in some cases, it is not feasible to select a temperature parameter that performs well throughout all the optimization steps. Since direct sampling is not possible due to not having access to the explicit form of the quantity we are sampling from, Boltzmann sampling relies on using Markov chain Monte Carlo (MCMC) methods [15], which enables efficient sampling in higher dimensions of unnormalized probabilities.

In this paper, we propose an alternative formulation of Boltzmann sampling by instead proposing to sample directly from the acquisition function, which avoids having to set a temperature parameter and has better performance across benchmarks. We analyze different MCMC methods found across the literature for performing acquisition sampling and evaluating the resulting Bayesian optimization performance. In particular, we study the

main challenges of applying MCMC to acquisition sampling: (1) we deal with a constrained domain defined by the search space of BO, (2) numerical instabilities of the acquisition function prevent proper, accurate representation of the target density, and (3) acquisition functions are highly multimodal, and sampling them is challenging for MCMC methods.

1.1. Bayesian Optimization

Bayesian optimization [16] is a sequential optimization method that aims to optimize expensive functions in a sample-efficient manner. It employs a *probabilistic surrogate model* $p(f)$ that learns the characteristics of the function that we aim to optimize, capturing the uncertainty and providing predictive distributions, which is then used by an *acquisition function* $\alpha(\mathbf{x}, p(f))$ that rates the next queries. Formally, it aims to optimize an unknown objective function $f : \mathcal{X} \to \mathbb{R}$ over a domain $\mathcal{X} \subset \mathbb{R}^d$, by carefully selecting the queries of the function to efficiently identify the optima $\mathbf{x}^*$. At iteration t, for readability, all previously observed values as $y_{1:t}$ at queried points $\mathbf{x}_{1:t}$ are used to construct a probabilistic surrogate model $\mathcal{M}_t = p(f|y_{1:t}, \mathbf{x}_{1:t})$. For readability, our notation will omit the subscript for observed data, defining $\mathbf{X} = \mathbf{x}_{1:t}$ and $\mathbf{y} = y_{1:t}$, with a set of points and values defined as $D_{1:t} = \{\mathbf{x}_{1:t}, y_{1:t}\}$.

This probabilistic surrogate model frequently takes the form of a Gaussian process (GP). It is a distribution over functions with a continuous domain, which is defined by $GP(\mu(\cdot), k(\cdot,\cdot))$, where $\mu : \mathcal{X} \to \mathbb{R}$ is the mean function and $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a covariance function or kernel that has kernel hyperparameters $\boldsymbol{\theta}$. For the experiments, we assume $\mu(\cdot) = 0$, and the kernel k is the Matérn kernel 5/2, defined as

$$k_{\text{Matérn}}(\mathbf{x}, \mathbf{x}') = \sigma_f \left(1 + \frac{\sqrt{5}|\mathbf{x} - \mathbf{x}'|}{\ell} + \frac{5(\mathbf{x} - \mathbf{x}')^2}{3\ell^2}\right) \exp\left(-\frac{\sqrt{5}|\mathbf{x} - \mathbf{x}'|}{\ell}\right) \tag{1}$$

with parameter ℓ as the lengthscale and the scaler parameter σ_f as the amplitude parameter. These are the hyperparameters of the GP that we will need to adjust $\boldsymbol{\theta} = \{\ell, \sigma_f\}$. Note that all methods in this paper are independent of mean and kernel choice.

Having decided both the mean $\mu(\cdot)$ and kernel $k(\cdot,\cdot)$ functions, we can now use the GP posterior model to make predictions at query points $\mathbf{x}_q$, which are normally distributed $\mathbf{y}_q \sim \mathcal{N}(\mu(\mathbf{x}_q), \sigma^2(\mathbf{x}_q))$. This predictive distribution can be computed in closed form:

$$\mu(\mathbf{x}_q) = \mu(\mathbf{x}_q) + \mathbf{k}(\mathbf{x}_q)^T \mathbf{K}^{-1} \mathbf{y} \tag{2}$$

$$\sigma^2(\mathbf{x}_q) = k(\mathbf{x}_q, \mathbf{x}_q) - k(\mathbf{x}_q, \mathbf{x})^T \mathbf{K}^{-1} k(\mathbf{x}_q, \mathbf{x}) \tag{3}$$

where the covariance matrix of the observed data points is $\mathbf{K} = k(\mathbf{X}, \mathbf{X})$ for noiseless observations, or, when modeling the noise in the observations $y = f(\mathbf{x}) + \epsilon$, the kernel becomes $\mathbf{K} = k(\mathbf{X}, \mathbf{X}) + \sigma_n^2 I$, with σ_n^2 being the Gaussian likelihood noise variance.

With the GP as our model $\mathcal{M}$, we can now define the overall algorithm for Bayesian optimization, as shown in Algorithm 1. First, it requires some initial data. Then, it involves sequentially updating the model $\mathcal{M}$ with the current observations, choosing a query with a selection strategy and performing the observation of said query. This is repeated until the budget T is exhausted, in which it returns the best-observed value $\{\mathbf{x}^*, y^*\}$. This is a valid recipe for sequential and batched parallel selection strategies, including asynchronous evaluations [9], which can be done by fantasizing about possible values of pending observations.

Algorithm 1 Bayesian Optimization

Require: Budget T
1: $D_{1:p} \leftarrow$ Query p initial points based on LHS
2: **for** $t = p \dots T$ **do**
3: $\quad \mathcal{M}_t \leftarrow$ Update surrogate model with $D_{1:t}$
4: $\quad \mathbf{x}_q \leftarrow$ Selection strategy using $\mathcal{M}_t$ ▷ See Section 2
5: $\quad D_{1:t+1} \leftarrow D_{1:t} \cup \{\mathbf{x}_q, f(\mathbf{x}_q) + \epsilon\}$ ▷ Observe query $\mathbf{x}_q$
6: **end for**
7: return $\leftarrow \{\mathbf{x}^*, y^*\} = D_i \quad \text{s.t.} \quad i = \arg\max_i \mathbf{y}_i$

Updating the model involves estimating the Gaussian process hyperparameters $\boldsymbol{\theta}$, typically by using the log marginal likelihood. The marginal likelihood or evidence $p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta})$ is the probability of the observed data given the input locations $\mathbf{X}$ and the hyperparameters $\boldsymbol{\theta}$. Log marginal likelihood is expressed as

$$\log p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}) = \underbrace{-\frac{1}{2}(\mathbf{y} - \mu(\mathbf{X}))^T \mathbf{K}^{-1}(\mathbf{y} - \mu(\mathbf{X}))}_{\text{Data fit}} \underbrace{-\frac{1}{2}\log|\mathbf{K}|}_{\substack{\text{Complexity} \\ \text{penalty}}} \underbrace{-\frac{n}{2}\log 2\pi}_{\text{Normalization}} \tag{4}$$

with the data term assessing how the model explains the observed data; the complexity penalty term discourages complexity that might overfit the data and normalization constant to produce a valid probability distribution. We will employ an *empirical Bayes* approach, where the hyperparameters are set to the most likely value, that is the maximization of the log marginal likelihood. This optimization can be done with numerical methods such as gradient descent or gradient-free methods such as L-BFGS.

Bayesian optimization requires some initial data $D_{1:p}$ so that we can start making predictions and select the next queries. Therefore, the optimization is initialized with p evaluations from a low discrepancy sequence, such as a Latin Hypercube sampling or a Sobol sequence.

1.2. Distributed BO

Most parallel versions of Bayesian optimization use a form of batch BO, where in each step, a set of query points are generated following a specific acquisition method that guarantees diversity and avoids repetitions on the query set [7,9,17,18]. As shown in Figure 1, these methods rely on a central node that has access to all the information and guarantees the diversity of query points. Furthermore, batch BO requires that queries are generated synchronously, which might be a problem in applications where different experiments might vary in time [9]. Instead, here we propose a fully distributed BO method as shown in Figure 1, which does not have any centralized node; all computations can be performed independently at each node within the distributed system, where diversity is built by construction. An advantage of this structure is that queries can be generated asynchronously as soon as new resources are available to perform experiments.

The BO algorithm from Algorithm 1 can be modified to a fully distributed system, as shown in Algorithm 2, where it shows the operations of a single node of the distributed system. Compared to the sequential BO algorithm, the only changes required are that the evaluation of initialization data is distributed across nodes, that nodes broadcast and receive their latest observations, and, most importantly, that the selection strategy is compatible with being fully distributed, that is, it ensures that the nodes are capable of observing different queries even if they have exactly the same data $D_{1:t}$ and model $\mathcal{M}$, such as in Thompson sampling [4,12] or Boltzmann sampling [13,14].

Algorithm 2 Distributed Bayesian Optimization Node

Require: Budget T
$D_{1:p/N} \leftarrow$ Query p/N initial points based on LHS
Broadcast $\leftarrow D_{1:p/N}$ ▷ To all other available nodes
for $t = p \dots T$ **do**
  $D_{1:t+1} \leftarrow D_{1:t} \cup \{\mathbf{x}_{\text{other}}, \mathbf{y}_{\text{other}}\}$ ▷ Collect other node data if available
  $\mathcal{M}_t \leftarrow$ Update surrogate model with $D_{1:t}$
  $\mathbf{x}_q \leftarrow$ Selection strategy using $\mathcal{M}_t$
  $D_{1:t+1} \leftarrow D_{1:t} \cup \{\mathbf{x}_q, f(\mathbf{x}_q) + \epsilon\}$
  Broadcast $\leftarrow \{\mathbf{x}_q, f(\mathbf{x}_q) + \epsilon\}$ ▷ To all other available nodes
end for
Sync data with other nodes ▷ So that all nodes return the same optimum
return $\leftarrow \{\mathbf{x}^*, \mathbf{y}^*\} = D_i$ s.t. $i = \arg\max_i \mathbf{y}_i$

Figure 1 shows how the nodes only need to broadcast their queries and observed values $\{\mathbf{x}_i, y_i\}$ to have access to all the information available at each moment, which requires minimal communication bandwidth. However, each node can perform optimization independently. Therefore, communication can be asynchronous and resilient to network failures, as the order of queries and observations does not affect the process, and nodes can keep working when isolated from the rest of the network if needed.

2. Selection Strategies

In this section, we present several acquisition function methods to generate new queries. Traditionally, acquisition maximization has been used in Bayesian optimization as the optimal decision method. However, we have seen how it can be difficult to parallelize and suffer from model bias. In contrast, acquisition sampling methods can be trivially parallelized, and they are more robust to model mismatch [14].

2.1. Acquisition Maximization

Traditionally, query selection in BO involves greedily optimizing an acquisition function $\alpha(\cdot)$ to select the next query $\mathbf{x}_{t+1} \in \mathcal{X}$, formally:

$$\mathbf{x}_{t+1} = \arg\max_{\mathbf{x} \in \mathcal{X}} \alpha(\mathbf{x}, p(f \mid \mathbf{y}_{1:t}, \mathbf{x}_{1:t})) \tag{5}$$

Acquisition functions are designed to trade off exploitation and exploration. Exploitation involves selecting queries that improve upon the current best or that have a high predictive value $\mu(\mathbf{x}_q)$. Exploration involves selecting queries that reduce model uncertainty or that have high uncertainties $\sigma^2(\mathbf{x}_q)$, respectively.

Common acquisition functions are simple utility functions, such as Upper Confidence Bound [19] and expected improvement (EI) [2]. Other acquisition functions follow a more information theoretic approach about the optimum for query selection, such as Entropy Search [20], Predictive Entropy Search [21], and Max-value Entropy Search [22], and typically involve more compute-intensive calculations.

2.1.1. Expected Improvement

In this paper, without the loss of generality to the choice of acquisition function, we will focus our analysis on EI [2], a popular choice for acquisition function since it provides good BO efficiency in practice while involving relatively fast computations since it relies on the posterior predictive μ and σ^2 to rate a query point $\mathbf{x}_q \in \mathcal{X}$:

$$EI_t(\mathbf{x}) = \mathbb{E}_{p(\mathbf{y}_{t+1} \mid \mathbf{y}_{1:t}, \mathbf{x}_{1:t})}[\max(0, \mathbf{y}_{t+1} - \rho_t)] = \mathrm{h}\left(\frac{\mu(\mathbf{x}) - \rho_t}{\sigma(\mathbf{x})}\right)\sigma(\mathbf{x}) \tag{6}$$

where $\rho_t = \max(\mathbf{y}_1, \ldots, \mathbf{y}_t)$ is the incumbent optimum at current iteration t and $\mathrm{h}(\mathbf{z}) = \phi(\mathbf{z}) + \mathbf{z}\Phi(\mathbf{z})$, with ϕ and Φ as the standard normal density and cumulative distribution function, respectively.

Acquisition function optimization is still challenging due to acquisition functions being non-convex. Furthermore, as more data are acquired into the GP, $EI_t(\mathbf{x})$ becomes highly multimodal, which can be seen in Figure2.

This is a challenge since optimization can be prone to get stuck in local minima. In practice, BO implementations circumvent this by using a multi-restart optimization approach, that is, multiple optimizations are performed from different initial points $\mathbf{x} \in \mathcal{X}$ with the hope of one reaching the mode of the global optimum.

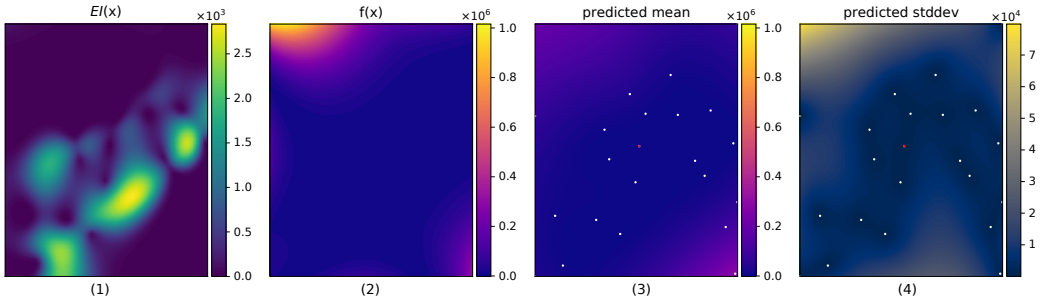

Figure 2. A 2D example showing the multimodality of the acquisition function in the left-most plot. From left to right, it shows (1) the expected improvement $EI(\mathbf{x})$, (2) the function $f(\mathbf{x})$, (3) the predictive mean of the GP $\mu(\mathbf{x})$, and (4) the predictive standard deviation of the GP $\sigma(\mathbf{x})$. Dots represent the observed locations.

Theoretically, since we use a GP, $EI(\mathbf{x})$ is computed with a Gaussian expectation, $p(\mathbf{y}|\mathbf{x}) \sim \mathcal{N}(\mu(\mathbf{x}_q), \sigma^2(\mathbf{x}_q))$, which has unbounded support, meaning that $EI(\mathbf{x})$ values and gradients are non-zero—although there are some edge cases, such as evaluating at data points locations with zero GP noise for stationary kernels—which is a useful property to help guide the optimization towards higher $EI(\mathbf{x})$ values. However, in practice, when $\mathbf{z} = (\mu(\mathbf{x}) - \rho_t)/\sigma(\mathbf{x})$ is small, meaning that the current model is very confident that there will be marginally small chances of improvement at $\mathbf{x}$, the $EI(\mathbf{x})$, EI values can become numerically zero. This is a problem when computing the logarithm, as can be seen in Figure 3.

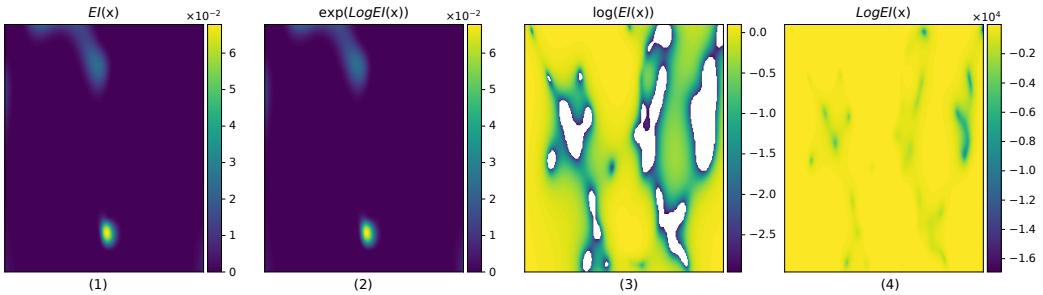

Figure 3. A 2D example problem to show the numerical instabilities of the expected improvement $EI(\mathbf{x})$. It shows different computations of the $EI(\mathbf{x})$ using a double-precision floating point. From left to right, it showcases the inaccuracy of $EI(\mathbf{x})$ compared to $LogEI(\mathbf{x})$: (1) shows the default $EI(\mathbf{x})$, (2) visually shows that $LogEI(\mathbf{x})$ computes the correct quantity of $EI(\mathbf{x}) \approx \exp(LogEI(\mathbf{x}))$, (3) shows the inaccuracies of $EI(\mathbf{x})$ when computing $log(EI(\mathbf{x}))$, with unusable white regions representing minus infinite values resulting from $EI(\mathbf{x})$ being numerically zero, and (4) shows that $LogEI(\mathbf{x})$ implementation is capable of representing $EI(\mathbf{x})$ values much lower than 1×10^{-3} without numerical issues.

To avoid this, we suggest using the *log expected improvement* function $LogEI(\mathbf{x})$ as presented in [23], which introduces a numerically stable implementation to the logarithm of $EI(\mathbf{x})$. As shown in Figure 3, this is more accurate compared to naively computing $\log(EI(\mathbf{x}))$. Since the optimum of $\log(EI(\mathbf{x}))$ is the same as that of the $EI(\mathbf{x})$, we can use it interchangeably to find the next query. $LogEI(\mathbf{x})$ can be computed as

$$LogEI(\mathbf{x}) = \log\left(h\left(\frac{\mu(\mathbf{x}) - \rho_t}{\sigma(\mathbf{x})}\right)\right) + \log(\sigma(\mathbf{x})) \tag{7}$$

where $\log(h(\cdot))$ can be computed in a numerically stable way with

$$\log(h(z)) = \begin{cases} \log(\phi(z) + z\Phi(z)) & z > -1 \\ -z^2/2 - c_1 + \text{log1mexp}(\log(\text{erfcx}(-z/\sqrt{2})|z|) + c_2 & 1/\sqrt{\epsilon} < z \leq -1 \\ -z^2/2 - c_1 - 2\log(|z|) & z \leq -1/\sqrt{\epsilon} \end{cases} \tag{8}$$

where $c_1 = log(2\pi)/2$ and $c_2 = \log(\pi/2)/2$ and ϵ is the numerical precision. log1mexp and erfcx are numerically stable implementations of $\log(1 - \exp(\mathbf{x}))$ and the *scaled complementary error function* $\exp(\mathbf{z}^2)\text{erfc}(\mathbf{z})$, respectively, and can be found in scientific software libraries, such as the Tensorflow Probability library [24] or SciPy [25].

2.1.2. Thompson Sampling

An alternative approach involves working directly with function values, such as sampling from the posterior distribution of the Gaussian process, drawing i.i.d. samples from the distribution of $\hat{f} \sim p(f|y_{1:t}, \mathbf{x}_{1:t})$, and using the maximizer as the next query point:

$$\mathbf{x}_{t+1} = \arg\max_{\mathbf{x} \in \mathcal{X}} \hat{f}(x) \tag{9}$$

This approach is known as Thompson sampling (TS), a strategy for selecting the next action in a multi-armed bandit problem but, in this case, adapted to the BO setting [4]. Similarly to other acquisition functions, TS is capable of leveraging the exploration and exploitation trade-off. The randomness in the drawn function means that points with high average value (exploitation) and points with high uncertainty (exploration) can yield high values and be the maximizers.

When sampling, $\hat{f}$ is infinite-dimensional; thus, generating exact samples is unfeasible. For computing exact samples, we can use a discretization over the domain $\mathcal{X}$ and evaluate samples at those points. However, sampling from a multivariate Gaussian distribution has $O(N^3)$ computational complexity due to requiring a Cholesky decomposition, which limits the granularity of the discretization. It is beyond the scope of this paper, but note that there are existing approaches to reduce the computation complexity, such as sparse GP approximations [26], leveraging the use of random Fourier features [27] to approximate kernel computations or enabling continuous samples using pathwise conditioning [28].

One of the advantages of TS is that we can parallelize the BO algorithm by drawing multiple functions $\hat{f}$ from the GP, which provides numerous maximizers that we can use to acquire parallel next queries [4].

2.2. Acquisition Sampling

Selection strategies such as the acquisition functions presented in Section 2.1.1 are spatially greedy as they only select the point that maximizes the acquisition function and ignores all information provided by the acquisition function across the optimization domain $\mathcal{X}$. This can be seen in Figure 4. Moreover, despite acquisition functions being designed to trade off exploration and exploitation to efficiently carry, they rely on the assumption

that the surrogate model *is good enough* to predict both the expected function value and its uncertainty accurately. Since, in practice, surrogate model parameters are learned as the BO progresses, model inaccuracies are common, especially at the beginning of the BO, which might impact the theoretical optimality of some acquisition functions.

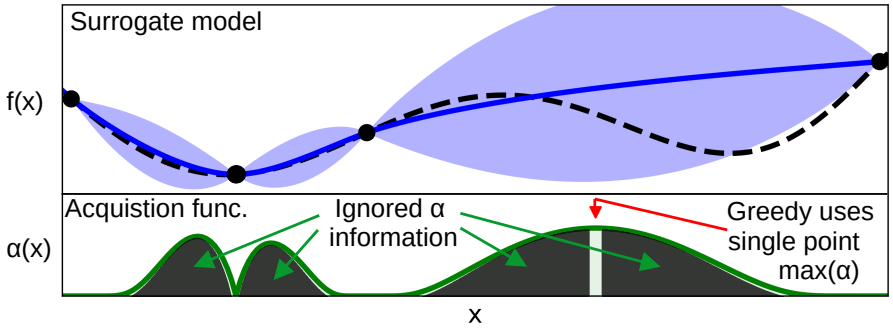

Figure 4. This figure shows a GP surrogate model (blue) plotted on top of an example 1D target function (black dashed). It allows us to showcase that the acquisition function (green) has information across all the input space and, by being greedy and selecting the maximum of the acquisition function (red arrow), we are ignoring a lot of information contained across the domain that can be as valuable as the maximum. Figure inspired by [13].

2.2.1. Boltzmann Sampling

One method that aims to use all the information provided by the acquisition function across the domain and improves model robustness is Boltzmann sampling [13], a sampling approach that uses the Boltzmann policy of the acquisition function:

$$p(\mathbf{x}_{t+1} \mid y_{1:t}, \mathbf{x}_{1:t}) = \frac{e^{\beta_t \alpha(\mathbf{x}_{t+1}, p(f \mid y_{1:t}, \mathbf{x}_{1:t}))}}{\int_{\mathbf{x} \in \mathcal{X}} e^{\beta_t \alpha(\mathbf{x}, p(f \mid y_{1:t}, \mathbf{x}_{1:t}))} d\mathbf{x}} \tag{10}$$

This defines a probability distribution for the next query $\mathbf{x}_{t+1} \sim p(\mathbf{x}_{t+1} \mid y_{1:t}, \mathbf{x}_{1:t})$. Based on temperature β_t, it allows the model to explore even if the model is completely biased and improves robustness to model inaccuracies [14]. As we will see in Section 3, some methods can work without normalizing the probabilities, which simplifies sampling the next query.

2.2.2. Direct Acquisition Sampling

Boltzmann sampling requires tuning a parameter for the temperature β_t. In practice, this means that this temperature must work throughout the entire Bayesian optimization. This can be challenging due to the variability of the resulting acquisition function as the Bayesian optimization progresses. If we use the EI as an example, at the beginning of BO, due to the sparsity of the data, we can expect higher improvements and thus higher EI values. As BO observes and gets closer to the global optimum, these improvements become smaller. This shows the variability of the EI. Inspired by Boltzmann, we propose carrying the sampling directly on the acquisition function $\alpha(\cdot)$ as an alternative, to avoid tuning β_t, so that the sampling probability becomes

$$p(\mathbf{x}_{t+1} \mid y_{1:t}, \mathbf{x}_{1:t}) = \frac{\alpha(\mathbf{x}_{t+1}, p(f \mid y_{1:t}, \mathbf{x}_{1:t})) - b}{\int_{\mathbf{x} \in \mathcal{X}} \alpha(\mathbf{x}, p(f \mid y_{1:t}, \mathbf{x}_{1:t})) d\mathbf{x}} \tag{11}$$

with b being a bias to ensure strictly positive p values to enable sampling when using non-negative acquisition functions. For the EI acquisition function that we will be using, we only need to set $b = 0$.

This formulation has good practical performance, as we will show in our experiments, without having to tune the β_t parameter. Additionally, a nice property of this formulation is that since we no longer have an exponential, we can introduce the $LogEI(\mathbf{x})$ formulation, which improves the numerical stability of $EI(\mathbf{x})$. This is possible due to sampling methods, such as MCMC, often working with ratios of probabilities, for which, to avoid numerical precision errors, computations are typically carried out using the logarithm of the probability [29]. This improves upon the original Boltzmann sampling [13] with the added benefits of the $LogEI(\mathbf{x})$ computations. Our target log-density probability that we will sample from becomes

$$\log p(\mathbf{x}_{t+1} \mid y_{1:t}, \mathbf{x}_{1:t}) \propto LogEI(\mathbf{x}_{t+1}, p(f \mid y_{1:t}, \mathbf{x}_{1:t})) \tag{12}$$

As we will see next, some sampling methods do not require the normalization of these log probability values.

3. MCMC for Acquisition Sampling

Sampling from an unknown probability typically is performed with a Markov chain Monte Carlo (MCMC) method [15]. The idea behind it is to generate a Markov chain $(\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_n)$ such that the stationary distribution of the chain is the target distribution $p(\mathbf{x})$, that is, as $n \to \infty$, it guarantees that $\mathbf{x}_n \sim p(\mathbf{x})$. In our case, the target distribution is the next query probability $p(\mathbf{x}) = p(\mathbf{x}_{t+1} \mid y_{1:t}, \mathbf{x}_{1:t})$, as defined in Equations (10) and (11).

MCMC implementations often operate with log densities directly to prevent floating point precision errors. This means that, for acquisition sampling, we can seamlessly introduce $LogEI(\mathbf{x})$, which helps in preventing numerically zero $EI(\mathbf{x})$ values and gradients.

Despite the diverse number of MCMC methods in the literature, sampling an acquisition function for BO presents a unique setting that is not typically found in MCMC applications. First, most acquisition functions are highly multimodal, with low probability areas between modes, as showcased in Figure 2. Second, sampling is performed in a constrained domain because BO operates within fixed bounds, typically a bounding box. In this work, we study different MCMC methods in the acquisition sampling setting and their impact on the resulting performance of the Bayesian optimization method.

3.1. Metropolis–Hastings

Metropolis–Hastings (MH) [30] is a classical MCMC method that performs a random walk combined with an acceptance–rejection sampling scheme. Given a current state $\mathbf{x}_n$, we want to generate $\mathbf{x}_{n+1}$ such that it satisfies the MCMC property. For that, it first generates a candidate $\mathbf{x}'$ from a *proposal distribution*, denoted as $q(\mathbf{x}'|\mathbf{x}_n)$. Then a ratio of acceptance,

$$r_a = \frac{p(\mathbf{x}')q(\mathbf{x}_n|\mathbf{x}')}{p(\mathbf{x}_n)q(\mathbf{x}'|\mathbf{x}_n)} \tag{13}$$

is used to accept or reject the candidate $\mathbf{x}'$. For $u \in [0, 1]$, generated using a uniform, it accepts the candidate $\mathbf{x}_{n+1} = \mathbf{x}'$ if $u \leq r_a$, otherwise, it rejects it by setting $\mathbf{x}_{n+1} = \mathbf{x}_n$. Note that, if the proposal q is symmetric, this simplifies to $r_a = p(\mathbf{x}')/p(\mathbf{x}_n)$.

For applying MH in a constrained domain, the general approach is to automatically reject the new candidate x' if it lies outside the domain $\mathcal{X}$, that is, we assume that the target probability outside the domain is zero. For dealing with the high multimodality of $\alpha(\cdot)$, we need to carefully select a proposal distribution q, since it needs to propose big transitions to jump between modes and smaller transitions so that the acceptance rate is high. If needed, q can be defined as mixture distribution, that enables more complex and adapted distributions.

3.2. Metropolis-Adjusted Langevin Algorithm

The Metropolis-adjusted Langevin algorithm (MALA) [31] takes steps using a discretized Langevin diffusion, defined by the stochastic differential equation

$$d\mathbf{x}_t = \frac{1}{2}\nabla_x \log(p(\mathbf{x}_t))dt + dB_t \tag{14}$$

where B denotes a d-dimensional Brownian motion, where in our case d corresponds to the problem dimensionality of the inputs. For discrete-time MCMC, a first-order Euler discretization is used:

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \frac{\epsilon^2}{2}\nabla_{\mathbf{x}} \log(p(\mathbf{x}_n)) + \epsilon u^n \tag{15}$$

where $u \sim \mathcal{N}(0, I)$ and ϵ is the integration step size. To correct first-order integration errors from discretization, the MH acceptance r_a is used. In this case, the proposal distribution is $q(\mathbf{x}_{n+1}|\mathbf{x}_n) \sim \mathcal{N}(\mu, \sigma^2)$, where $\mu = \mathbf{x}_n + \frac{\epsilon^2}{2}\nabla_{\mathbf{x}} \log(p(\mathbf{x}_n))$ and $\sigma^2 = \epsilon^2$.

3.3. Hamiltonian Monte Carlo

Hamiltonian Monte Carlo (HMC) [32] is a particular case of MH that uses the gradients of the density function to improve the acceptance ratio. It uses an approximate Hamiltonian dynamics simulation with leapfrog numerical integration alongside a Metropolis acceptance step. The idea is to simulate the trajectory of a particle in a potential field defined by the target distribution. It introduces an auxiliary momentum variable ρ to the state and draws from a joint density

$$p(\rho, \mathbf{x}) = p(\rho|\mathbf{x})p(\mathbf{x}) \tag{16}$$

that defines the following Hamiltonian

$$H(\rho, \mathbf{x}) = -\log p(\rho|\mathbf{x}) - \log p(\mathbf{x}) \tag{17}$$

which represents the total energy of the system H, with the resulting two log terms often described as the kinetic and potential energy of the system, respectively.

New state candidate $(\rho', \mathbf{x}')$ is generated by first drawing momentum $\rho' \sim \mathcal{N}(0, M)$, where M is the mass matrix, and then $(\rho', \mathbf{x}_n)$ is run under Hamilton's equations. To solve these motion equations, a symplectic integrator is needed, a numerical integrator for Hamiltonian systems that preserves the Hamiltonian energy. Furthermore, being a particular case of MH, it needs a time-reversible integrator. We will use a leapfrog integrator [15], which is a second-order time-reversible and symplectic integrator. Given a time discretization step size ϵ, it simulates the exact trajectory as

$$\rho_{t+\epsilon/2} = \rho_t - \frac{\epsilon}{2}\nabla \log p(\mathbf{x}_t) \tag{18}$$

$$\mathbf{x}_{t+\epsilon} = \mathbf{x}_t + \epsilon\rho_{t+\epsilon/2} \tag{19}$$

$$\rho_{t+\epsilon} = \rho_{t+\epsilon/2} - \frac{\epsilon}{2}\nabla \log p(\mathbf{x}_{t+\epsilon}) \tag{20}$$

which updates the momentum for half a step $\rho_{t+\epsilon/2}$, then performs a full step update of position $\mathbf{x}_{t+\epsilon}$ using the updated momentum and, finally, the momentum final half step $\rho_{t+\epsilon}$ is updated using the new position. At the end of the leapfrog integration steps, it generates a candidate $(\rho', \mathbf{x}')$ that is subjected to an MH acceptance such that $r_a = \exp(H(\rho, \mathbf{x}_n) - H(\rho', \mathbf{x}'))$.

3.4. No U-Turn Sampler

No U-turn sampler (NUTS) [33] is an HMC method that performs multiple steps informed by first-order gradient information but, as opposed to HMC, it uses a "no U-turn" criterion when sampling trajectories, that is, if the trajectory loops back and begins retracing itself, it stops expanding the trajectory. This improves efficiency by avoiding unnecessary computations and enables an adaptive trajectory length.

3.5. Cyclical SGLD

Stochastic gradient Langevin dynamics (SGLD) is a sampling method that combines stochastic gradient descent (SGD) and Langevin dynamics to achieve faster chain convergence. It uses stochastic gradient descent to approximate the gradient calculation; where there is no closed form, it is estimated from a dataset. In this case, we could use SGD to compute the gradient of the GP with a minibatch of the dataset, which could result in a reduced computation load. However, for comparison, we have decided to use the closed-form derivative of the full GP as in the other methods. Thus, the method becomes similar to the MALA. Another advantage of SGLD is that it uses a schedule on the step size such that $\sum_{n=1}^{\infty} \epsilon_n = \infty$ and $\sum_{n=1}^{\infty} \epsilon_n^2 < \infty$, which guarantees convergence in probability without the need for the MH acceptance r_a, as in the MALA.

Furthermore, for multimodal distributions, SGLD can be extended to allow the chain to move between different modes while maintaining the same convergence capabilities. This is called cyclical stochastic gradient Langevin descent (CyclicalSGLD) [34], which extends SGLD by introducing warm restart cycles. For each cycle, it restarts the step size to an initial value and then lowers the step size as discussed before. This process alternates between exploration and sampling stages within each cycle, as shown in Figure 5. Exploration involves stochastic gradient descent (SGD), with a larger step size allowing jumping between modes of the distribution. The sampling stage involves SGLD with a lower step size to properly characterize the local density of the target distribution.

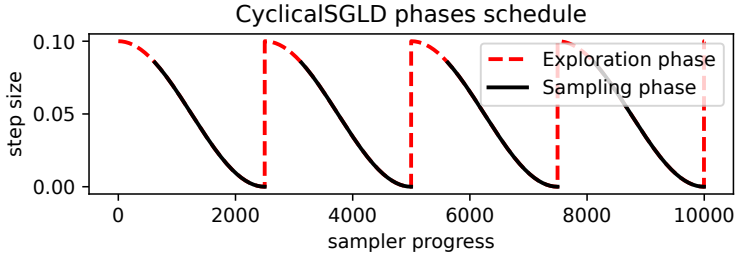

Figure 5. CyclicalSGLD schedules the parameter step size in cycles as the sampler progresses. It shows its two distinct phases: exploration using SGD and sampling using SGLD. Figure inspired by Blackjax's *The Sampling Book Project* [35].

4. Results

4.1. Methodology

We will evaluate the methods presented in previous sections: *expected improvement maximization* (MaxEI), *Thompson sampling* (TS), and *direct acquisition sampling* (AS). The AS is broken down into different MCMC sampling methods that we evaluate. The analysis will be performed for parallel queries similar to [13], that is, we perform multiple queries per GP update for TS and AS methods. We have not included *Boltzmann sampling* (BS) in the overall comparison as the performance was consistently worse than AS for all the experiments, as shown in Section 4.2.3. Both AS and BS use the expected improvement as the underlying acquisition function.

All methods are implemented using Python 3.10 [36], Jax 0.4.27 [29], and Blackjax 1.2.3 [35] for MCMC implementations and tinygp 0.3.0 [37] for Gaussian processes. Methods relying on gradients use automatic differentiation and parallel queries are computed using automatic vectorization.

All methods use a Gaussian process (GP) with a Matérn kernel with $\nu = 5/2$ and additive Gaussian noise $\sigma_n = 1 \times 10^{-3}$.

Initial hyperparameters of the GP are the amplitude scaler $\sigma_f = 0.1$, lengthscale $l = 1.0$, and mean $\mu = 0$, which are then updated by optimizing the log marginal likelihood of the GP. Starting from 10 initial points, each method performs 150 observations. All methods use common random numbers, and experiments are repeated 10 times. Internally, the input domain for each benchmark is normalized to $[0, 1]^d \in \mathbb{R}$, which alleviates the need to choose different parameters for initial PG and method parameters across the different functions.

MaxEI and AS methods use the $LogEI(\mathbf{x})$ numerically stable computation [23]. TS and AS will propose a batch of five queries since its main application is in parallelization and following the experimental setup of [13].

For AS methods, we will evaluate each of the MCMC algorithms introduced within Section 2.2.1. All methods use chains of 4000 length. Note that, in AS methods, the chain length is not equivalent to the number of acquisition evaluations since methods can use multiple chains or perform multiple evaluations for the leapfrog integrator.

The specific parameters for each method are detailed next:

1. Hamiltonian Monte Carlo (HMC) uses a step size of 1×10^{-2}, with five integration steps and identity mass matrix $M = I^D$.
2. The Metropolis-adjusted Langevin algorithm (MALA) uses a step size of 1×10^{-2}.
3. No U-turn sampler (NUTS) uses a step size of 1×10^{-2} and identity mass matrix $M = I^D$.
4. CyclicalSGLD uses 30 cycles, alternating exploration and sampling, with a 25–75% ratio, respectively, an initial step size of 1×10^{-3}, and an SGD learning rate of 1×10^{-4}.
5. Mixture distribution Metropolis–Hastings (MMH) uses a proposal that combines different sizes of normal distributions:

$$q(\mathbf{x}_n) = \frac{1}{4}\Big(\mathcal{N}(\mathbf{x}_n, 0.01) + \mathcal{N}(\mathbf{x}_n, 0.1) + \mathcal{N}(\mathbf{x}_n, 0.3) + \mathcal{U}^d(0, 1)\Big) \tag{21}$$

which can be sampled following a stratified sampling strategy:

$$\mathbf{x}_n \sim \begin{cases} \mathcal{N}(\mathbf{x}_n, 0.01) & u < 0.25 \\ \mathcal{N}(\mathbf{x}_n, 0.1) & 0.25 < u \leq 0.5 \\ \mathcal{N}(\mathbf{x}_n, 0.3) & 0.5 < u \leq 0.75 \\ \mathcal{U}^d(0, 1) & 0.75 < u \leq 1.0 \end{cases} \tag{22}$$

where $u \sim \mathcal{U}(0, 1)$ is the selecting variable. This proposal achieves a good balance between modeling the locality of the density and jumping between different modes of the acquisition function, considering that the input space is normalized.

To acquire N queries from MCMC, there are two practical approaches found in the literature. One involves maintaining a single long chain and uniformly selecting N samples from the chain [38]. The other approach involves running multiple independent chains [39], where the chains can be used for convergence analysis, and allowing the chains to be run in parallel. For our implementation, we will rely on the second approach by running N independent chains and taking a single sample after a burn-in period of 4000 samples in the chain. The reason is that we can use automatic vectorization to parallelize both the chains and the computations of the target probabilities.

To help prevent any MCMC method from going out of bounds, we drew inspiration from [40], which proposes an unbounded BO with bounds dynamically increasing as needed. To reflect the current boundary in the acquisition function, they put a smooth weighting function over the acquisition function that tends to zero as it approaches the current boundary. In a similar fashion, we propose to smooth the bounds by using

$$w(\mathbf{x}) = a \exp \frac{-\mathbf{x}^2}{2c^2} \tag{23}$$

with parameters $a = 1$ and $c = 1 \times 10^{-2}$; $w(\mathbf{x})$ can be used to smooth a step function using the bounds of the optimization search, which maintains derivability and helps the gradient to point back towards the search domain, which is useful to prevent MCMC methods from going out of bounds. Figure 6 shows a step function and its equivalent bump function using $w(\mathbf{x})$.

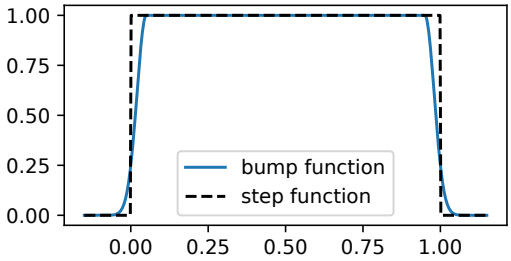

Figure 6. Comparison between a step function and its smoothed-out counterpart using a bump function in the domain $(0, 1)$. The advantage of the bump function is that it is derivable and the resulting gradients point back towards the optimization domain. Note that for this visualization, the smooth effect is wider for illustrative purposes. In practice, the width of the smoothness is tighter.

4.2. Experiments

Experiments are carried out using benchmark functions typically used in Bayesian optimization [41]. This allows us to know the global optima to better assess the performance of different selection methods and, in particular, the different MCMC methods. The figures show the progress of the incumbent, that is, the current best point found at each optimization step t. Since experiments are run multiple times, in the plots, we show the average and a shaded region corresponding to two times the standard deviation of the incumbents across the 10 repetitions. For ease of visualization, we also show the logarithm of the incumbent with respect to the global minima, helping discern the best-performing methods visually. In addition, the legend of each plot is sorted in terms of the performance ranking.

In the following subsections, we study experiments performed on benchmark functions, such as a smooth valley-shaped 2D function and multiple N-dimensional functions, and a real robot problem of an autonomous rover finding the optimal trajectory in uneven terrain.

4.2.1. Benchmark Functions

The Rosenbrock function [42] is a 2D problem where the optimum lies within a very plain valley. Finding this valley with exploration is easy; however, identifying the global solution is harder. It has the following equation:

$$f_{\text{Rosenbrock}}(\mathbf{x}) = 100(x_2 - x_1^2)^2 + (x_1 - 1)^2 \tag{24}$$

with the function being evaluated within the domain $\{x_1, x_2\} \in \{[-0.5, 3], [-1.5, 2]\}$. As shown in Figure 7, it is easy to explore with BO and arrive at the valley, as shown by all

methods finding good values very quickly. However, if we look at the logarithmic, it shows the improvements during the exploitation, with MMH finding more refined solutions.

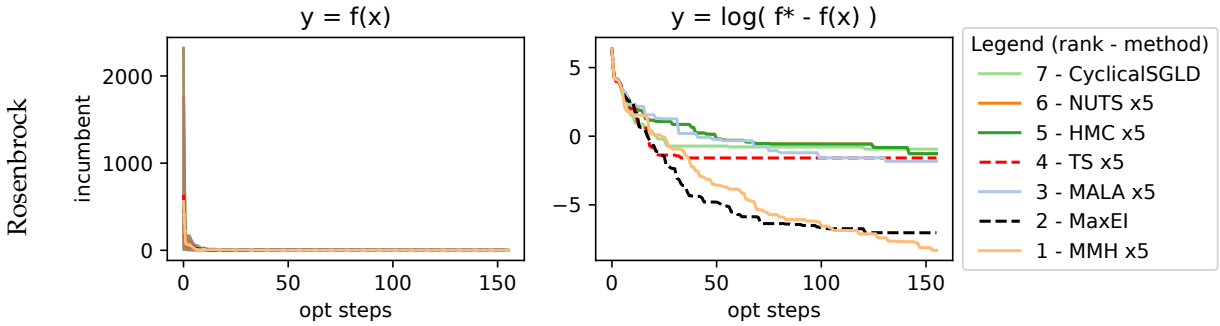

Figure 7. Results on 2D Rosenbrock function. It is a simple function to explore and quickly acquire good-enough values, but the finer exploitation towards optimum is harder, as shown in the log plot. MMH, despite its simplicity, is capable of being as performant as a sequential MaxEI.

We also evaluate using N-dimensional benchmark functions, that is, functions that are also parametrized by the number of dimensions d. This allows us to study the behavior of different methods as dimensionality increases. In particular, we will focus on functions showcasing many local minima, such as Ackley and Alpine.

The Ackley function [42] is a function that contains many local minima. It has the following equation:

$$f_{\text{Ackley}}(\mathbf{x}) = -a \exp\left(-b\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}\right) - \exp\left(\frac{1}{d}\sum_{i=1}^{d}\cos cx_i\right) + a + \exp(1) \tag{25}$$

with recommended variables $a = 20$, $b = 0.2$, and $c = 2\pi$ and evaluated within the domain $x_i \in [-32.768, 32.768]$, and we evaluated it for the 3D, 5D, and 10D cases. Results in Figure 8 show that as dimensions increase, the performance of MaxEI and TS deteriorates compared to AS. Within the AS methods, MMH, CyclicalSGLD, and MALA methods show better performance.

We also evaluate the Alpine family of functions [41], which are two functions that also have many local minima. They have the following equations:

$$f_{\text{Alpine1}}(\mathbf{x}) = \sum_{i=1}^{d} |x_i \sin(x_i) + 0.1x_i| \tag{26}$$

$$f_{\text{Alpine2}}(\mathbf{x}) = \prod_{i=1}^{d} \sqrt{x_i}\sin(x_i) \tag{27}$$

with the search space being $x_i \in [-10, 10]$ and $x_i \in [1, 10]$ respectively. Alpine1 evaluation at 5D and 10D in Figure 9 shows how MaxEI performs better. For AS methods, only CyclicalSGLD matches the performance of MaxEI. MMH is the second, followed by the MALA. However, it seems that methods are stuck in suboptimal regions. This can be seen in Alpine2, evaluated at 5D and 10D, shown in Figure 10, which has a smaller search space and is able to reach better incumbent values. In this case, the best-performing method is CyclicalSGLD, followed by the MALA and MMH.

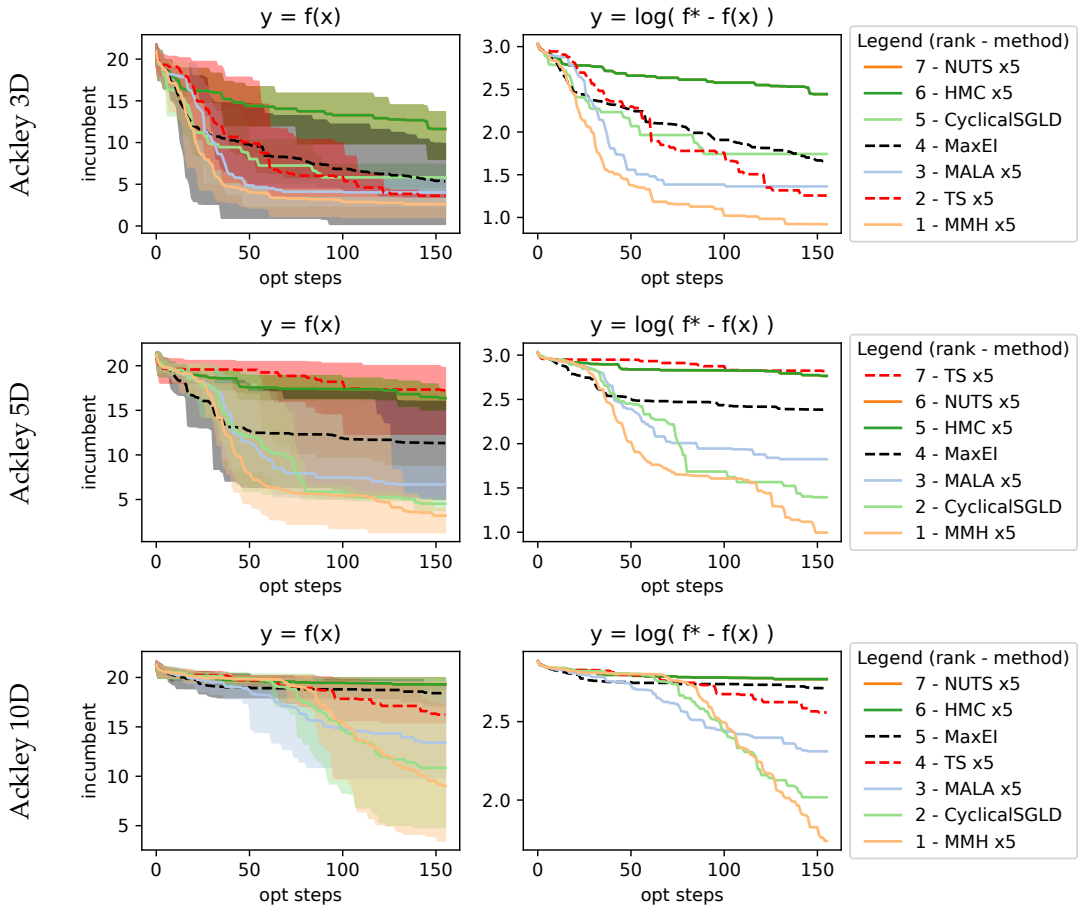

Figure 8. Results on 3D, 5D, and 10D Ackley function. As dimensions increase, optimizing the function becomes more of a challenge; as we can see, sampling methods perform better in higher dimensions.

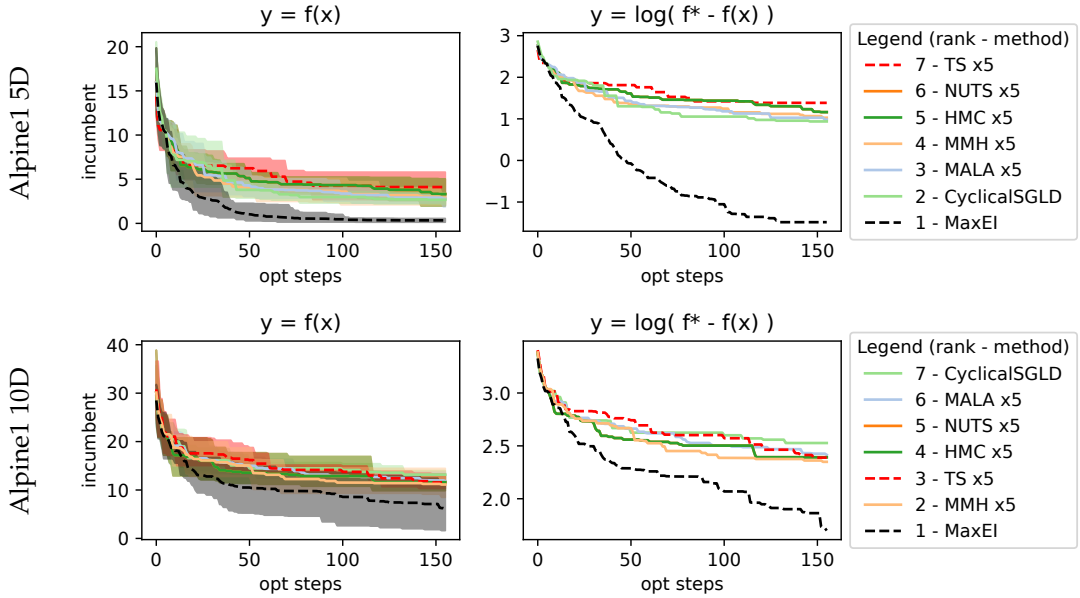

Figure 9. Results on 5D and 10D Alpine1 function. Most sampling methods perform similarly, with minor differences. Sequential maximization in MaxEI performs much better than the rest of the methods due to operating on a 5-query parallelization, which, in theory should always be suboptimal compared to a sequential approach.

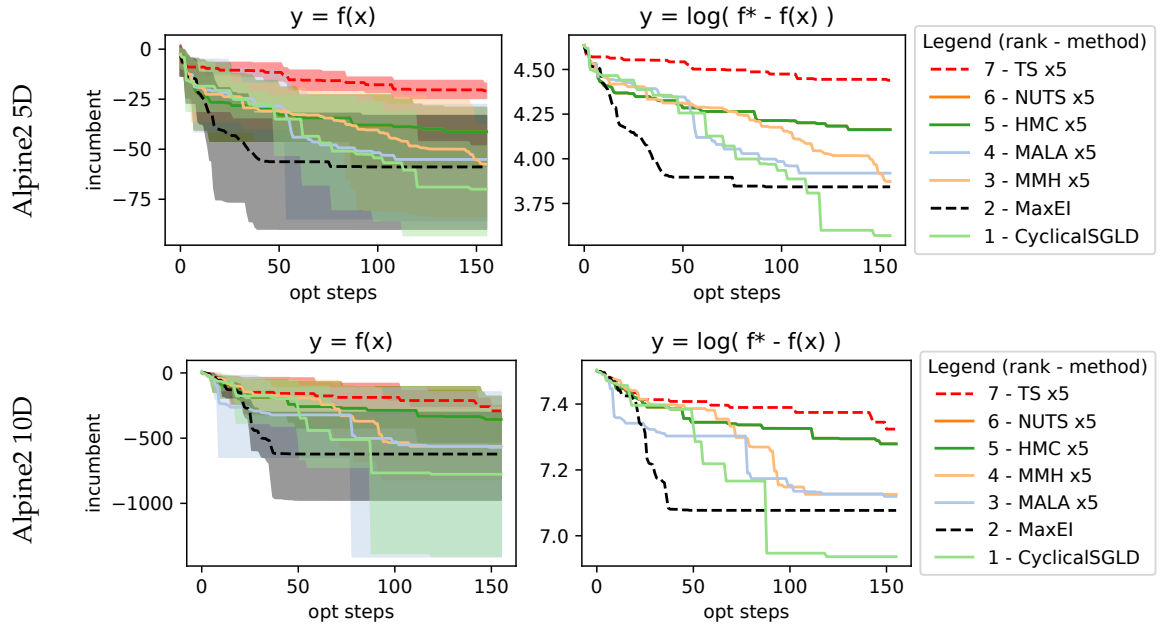

Figure 10. Results on 5D and 10D Alpine2 function. This function shows acquisition sampling capable of exploring much better than function sampling done by TS.

4.2.2. Rover Trajectory Problem

In this section, we cover the problem of finding the lowest-cost trajectory of a rover in a surface map between an initial point and an endpoint, where the cost represents possible terrain irregularities or obstacles that we wish to avoid. This setting is based on *optimizing rover trajectories* from Wang et al. [43], and we follow the changes introduced in [14]. We define a parametrized trajectory with a B-spline that starts fixed at the starting position, ends fixed at the goal position, and, as parameters, uses two control points that will define the curvature of the B-spline. The overall dimensionality of the problem we are solving is 4D since we have to optimize the two control points that are 2D. Trajectory cost is integrated along the trajectory at a fixed interval using 1000 steps. We use two different maps, as shown in the picture inside Figure 11, with both maps sharing the same start and goal but differing on the costs (e.g., different obstacles). Results are also shown in Figure 11, where it shows that MaxEI has some trouble finding the optimal zero-cost trajectory. Meanwhile, AS and TS are capable of finding such a trajectory. In particular, MMH and the MALA are the AS methods that are capable of solving the problem.

4.2.3. Ablation on Boltzmann Sampling

Here we compare Boltzmann sampling (BS) with different temperatures to direct acquisition sampling (AS). We compared them using the same MMH sampling method, which we chose since it was the best-performing MCMC method overall. The results in Figure 12 showcase how difficult is to choose a good fixed temperature for BS. We can clearly see that there is a point in the BO wherein the BS methods underperform, which aligns with our hypothesis that a single fixed temperature is not capable of working properly throughout the optimization, whereas our proposed approach AS circumvents this limitation. We only show results on the Ackley function, but the results were consistent in the other benchmarks.

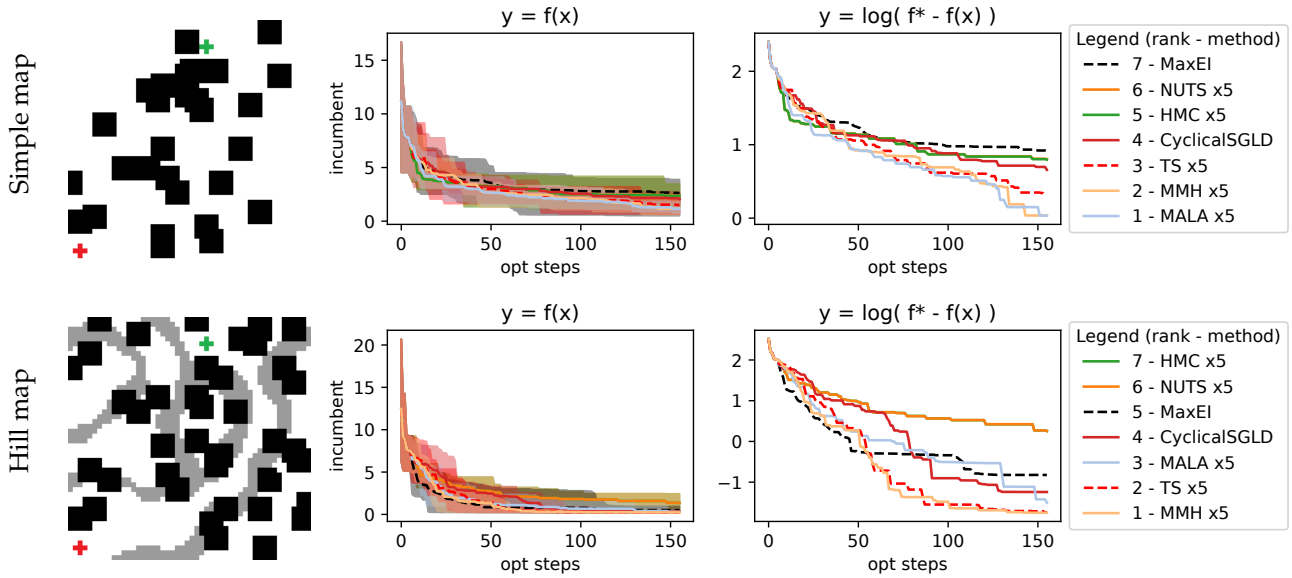

Figure 11. Results on the rover trajectory problem, showing a simple map and a map with hills. The starting position of the trajectory is from the red cross at the lower left, and the goal is the green cross at the top side. The cost of traversing a terrain is determined by how dark the space is in the picture, with zero cost in white. Results show that AS and TS are capable of finding the zero-cost trajectory, with MMH and the MALA clearly showing better performance than all the AS methods.

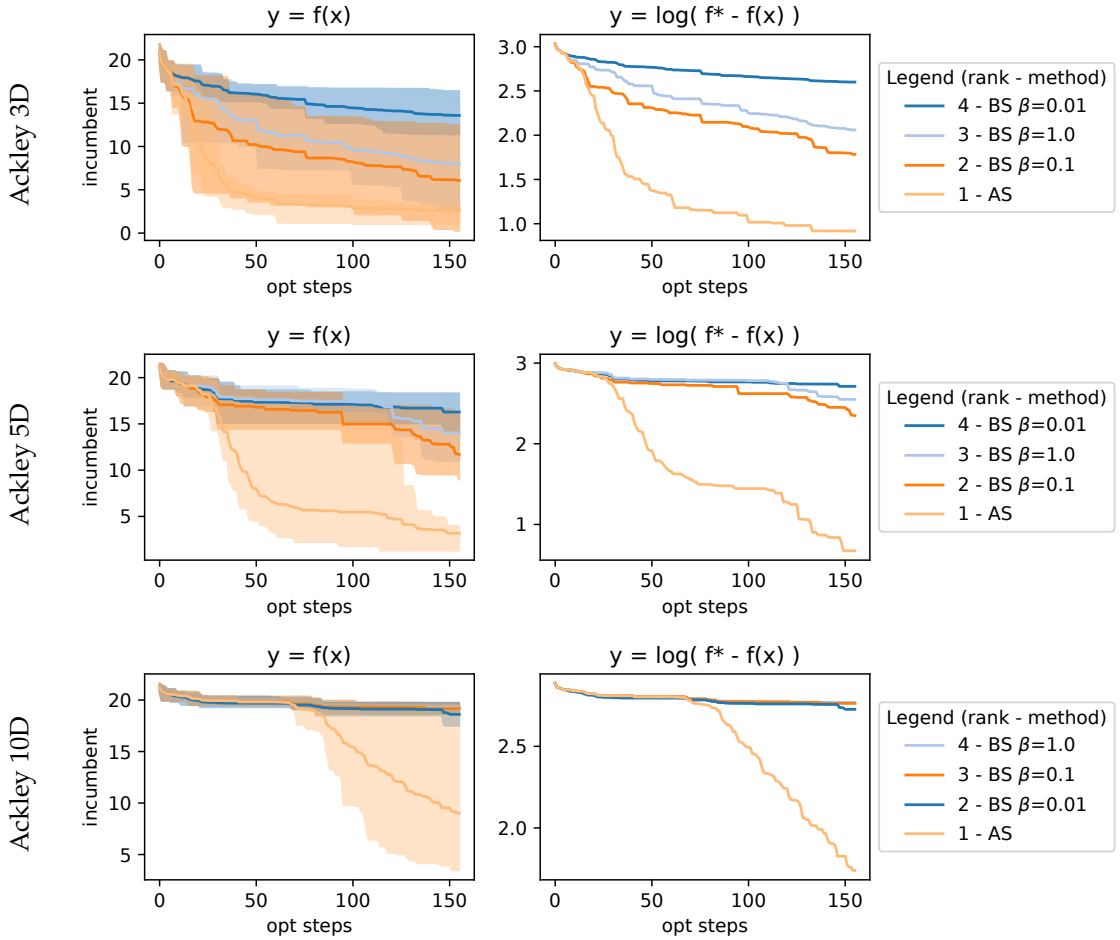

Figure 12. Results on 3D, 5D, and 10D Ackley function using MMH comparing AS against BS with different temperatures. These plots showcase the difficulty of tuning the β temperature parameter for BS. It shows how AS performs better without a parameter.

5. Discussion

The proposed experiments focused on studying the different MCMC methods when sampling from the acquisition function. Overall, out of the methods that we evaluate, CyclicalSGLD, MMH, and the MALA are the only methods that consistently rank on top.

The most underperforming methods, NUTS and HMC, suffer from one of the challenges of acquisition sampling, that is, the variability in the function that we sample as the BO progresses and its relationship to the MCMC parameters. In particular, if the mass matrix M is a poor approximation of the target distribution covariance, it will be required to compensate for other parameters to perform a more fine-grained integration (i.e., smaller step size ϵ and a higher number of integration steps) to maintain numerical precision. Even when actively tuning these parameters, it is still challenging [44]. In our setup, failing to choose good parameters across all Bayesian optimization steps leads to poor performance. Out of the best-performing methods, CyclicalSGLD and the MALA share that they use the gradient information, helping to improve MCMC efficiency, and the parameters are not as critical for chain convergence or hard to choose. Finally, the most interesting result is the MMH. Metropolis–Hastings is a classic algorithm that is also quite simple compared to other methods since it only has a single parameter: the proposal distribution. In our experiments, we opted for a mixture of distributions as a proposal, with a uniform distribution allowing global jumps to change between multiple modes of the acquisition function and multiple Gaussian processes that enable traversing the locality of the probability. Experiments show that, even on higher dimensions, this mixture approach is performant.

Regarding the bounded problems, our bump function helps to keep the chain within the search space of BO. However, with high gradients, the chain might jump out of bounds directly and reject many states of the MCMC chain. As a takeaway, we propose MMH as the better alternative, since we can reject jumps out-of-bounds, avoiding the problems associated with the previous methods.

Future research of this work will involve the bounded setting of MCMC methods, which, in the context of MCMC, is quite limited in the literature, especially if we focus on highly multimodal densities. In particular, research will include modifying good-performing methods, such as the MALA and CyclicalSGLD, to include bounds or complement with methods that actively deal with bounded domains, such as hit-and-run methods [45]. Additionally, automatic tuning of the MCMC parameters using the initial N samples as burn-in could be used to improve some of the methods' overall performance or adapt the parameters using an inner loop of Bayesian optimization [44].

Author Contributions: Conceptualization, J.G.-B. and R.M.-C.; methodology, J.G.-B. and R.M.-C.; software, J.G.-B.; validation, J.G.-B.; investigation, J.G.-B.; visualization, J.G.-B.; writing—original draft preparation, J.G.-B.; writing—review and editing, R.M.-C.; supervision, R.M.-C.; funding acquisition and resources, R.M.-C. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Spanish government projects PID2021-125209OB-I00 and TED2021-131150B-I00 (MCIN/AEI/10.13039/501100011033 and NextGenerationEU/PRTR) and the Aragon Government DGA T45_23R.

Data Availability Statement: The data presented in this study are openly available on GitHub at https://github.com/jgbarcos/better_bo_sampling, accessed on 7 January 2025.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

BO	Bayesian optimization
GP	Gaussian process
EI	Expected improvement
LogEI	Log expected improvement
MaxEI	Maximization expected improvement
TS	Thompson sampling
BS	Boltzmann sampling
AS	Direct acquisition sampling
MCMC	Markov chain Monte Carlo
MH	Metropolis–Hastings
MMH	Mixture distribution Metropolis–Hastings
MALA	Metropolis-adjusted Langevin algorithm
HMC	Hamiltonian Monte Carlo
NUTS	No U-turn sampler
SGLD	Stochastic gradient Langevin dynamics
SGD	Stochastic gradient descent

References

1. Jones, D.; Schonlau, M.; Welch, W. Efficient Global Optimization of Expensive Black-Box Functions. *J. Glob. Optim.* **1998**, *13*, 455–492. [CrossRef]
2. Mockus, J.; Tiesis, V.; Zilinskas, A. The application of Bayesian methods for seeking the extremum. In *Towards Global Optimisation 2*; Elsevier: Amsterdam, The Netherlands, 1978; pp. 117–129.
3. Wang, X.; Jin, Y.; Schmitt, S.; Olhofer, M. Recent Advances in Bayesian Optimization. *ACM Comput. Surv.* **2023**, *55*, 287. [CrossRef]
4. Hernandez-Lobato, J.; Requeima, J.; Pyzer-Knapp, E.; Aspuru-Guzik, A. Parallel and Distributed Thompson Sampling for Large-scale Accelerated Exploration of Chemical Space. In Proceedings of the ICML 2017, Sydney, NSW, Australia, 6–11 August 2017; pp. 1470–1479.
5. Ma, H.; Zhang, T.; Wu, Y.; Calmon, F.P.; Li, N. Gaussian Max-Value Entropy Search for Multi-Agent Bayesian Optimization. In Proceedings of the 2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Detroit, MI, USA, 1–5 October 2023; pp. 10028–10035. [CrossRef]
6. Kandasamy, K.; Vysyaraju, K.R.; Neiswanger, W.; Paria, B.; Collins, C.R.; Schneider, J.; Poczos, B.; Xing, E.P. Tuning hyperparameters without grad students: Scalable and robust bayesian optimisation with dragonfly. *J. Mach. Learn. Res.* **2020**, *21*, 1–27.
7. González, J.; Dai, Z.; Hennig, P.; Lawrence, N. Batch bayesian optimization via local penalization. In Proceedings of the AISTATS 2016, Cadiz, Spain, 9–11 May 2016; pp. 648–657.
8. Desautels, T.; Krause, A.; Burdick, J. Parallelizing exploration-exploitation tradeoffs in Gaussian process bandit optimization. *J. Mach. Learn. Res.* **2014**, *15*, 3873–3923.
9. Snoek, J.; Larochelle, H.; Adams, R. Practical Bayesian Optimization of Machine Learning Algorithms. In Proceedings of the NIPS 2012, Lake Tahoe, NV, USA, 3–8 December 2012; pp. 2960–2968.
10. Dai, Z.; Low, B.K.H.; Jaillet, P. Differentially Private Federated Bayesian Optimization with Distributed Exploration. In Proceedings of the Advances in Neural Information Processing Systems, Virtual, 6–14 December 2021; Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., Vaughan, J.W., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2021; Volume 34, pp. 9125–9139.
11. Egelé, R.; Guyon, I.; Vishwanath, V.; Balaprakash, P. Asynchronous Decentralized Bayesian Optimization for Large Scale Hyperparameter Optimization. In Proceedings of the 2023 IEEE 19th International Conference on e-Science (e-Science), Limassol, Cyprus, 9–13 October 2023; pp. 1–10. [CrossRef]
12. Kandasamy, K.; Krishnamurthy, A.; Schneider, J.; Poczos, B. Parallelised Bayesian Optimisation via Thompson Sampling. In Proceedings of the AISTATS 2018, Playa Blanca, Lanzarote, 9–11 April 2018.
13. Garcia-Barcos, J.; Martinez-Cantin, R. Fully Distributed Bayesian Optimization with Stochastic Policies. In Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19. International Joint Conferences on Artificial Intelligence Organization, Macao, Hong Kong, 10–16 August 2019; pp. 2357–2363. [CrossRef]
14. Garcia-Barcos, J.; Martinez-Cantin, R. Robust Policy Search for Robot Navigation. *IEEE Robot. Autom. Lett.* **2021**, *6*, 2389–2396. [CrossRef]

15. Brooks, S.; Gelman, A.; Jones, G.; Meng, X.L. *Handbook of Markov Chain Monte Carlo*; CRC press: Boca Raton, FL, USA, 2011.
16. Shahriari, B.; Swersky, K.; Wang, Z.; Adams, R.; de Freitas, N. Taking the human out of the loop: A review of Bayesian optimization. *Proc. IEEE* **2016**, *104*, 148–175. [CrossRef]
17. Ginsbourger, D.; Le Riche, R.; Carraro, L. Kriging is well-suited to parallelize optimization. In *Computational Intelligence in Expensive Optimization Problems*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 131–162.
18. Contal, E.; Buffoni, D.; Robicquet, A.; Vayatis, N. Parallel Gaussian process optimization with upper confidence bound and pure exploration. In Proceedings of the ECMLKDD 2013, Prague, Czech Republic, 22–26 September 2013; Springer: Berlin/Heidelberg, Germany, 2013; pp. 225–240.
19. Srinivas, N.; Krause, A.; Kakade, S.; Seeger, M. Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design. In Proceedings of the ICML 2010, Haifa, Israel, 21–24 June 2010.
20. Hennig, P.; Schuler, C. Entropy Search for Information Efficient Global Optimization. *J. Mach. Learn. Res.* **2012**, *13*, 1809–1837.
21. Hernandez-Lobato, J.; Hoffman, M.; Ghahramani, Z. Predictive Entropy Search for Efficient Global Optimization of Black-box Functions. In Proceedings of the NIPS 2014, Montreal, QC, Canada, 8–13 December 2014; pp. 918–926.
22. Wang, Z.; Jegelka, S. Max-value entropy search for efficient Bayesian optimization. In Proceedings of the International Conference on Machine Learning. PMLR 2017, Sydney, NSW, Australia, 6–11 August 2017; pp. 3627–3635.
23. Ament, S.; Daulton, S.; Eriksson, D.; Balandat, M.; Bakshy, E. Unexpected Improvements to Expected Improvement for Bayesian Optimization. In Proceedings of the Advances in Neural Information Processing Systems, New Orleans, LA, USA, 10–16 December; Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., Levine, S., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2023; Volume 36, pp. 20577–20612.
24. Dillon, J.V.; Langmore, I.; Tran, D.; Brevdo, E.; Vasudevan, S.; Moore, D.; Patton, B.; Alemi, A.; Hoffman, M.; Saurous, R.A. TensorFlow Distributions. *arXiv* **2017**, arXiv:1711.10604.
25. Virtanen, P.; Gommers, R.; Oliphant, T.E.; Haberland, M.; Reddy, T.; Cournapeau, D.; Burovski, E.; Peterson, P.; Weckesser, W.; Bright, J.; et al. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nat. Methods* **2020**, *17*, 261–272. [CrossRef]
26. Vakili, S.; Moss, H.; Artemev, A.; Dutordoir, V.; Picheny, V. Scalable Thompson Sampling using Sparse Gaussian Process Models. In Proceedings of the Advances in Neural Information Processing Systems, Virtual, 6–14 December 2021; Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., Vaughan, J.W., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2021; Volume 34, pp. 5631–5643.
27. Rahimi, A.; Recht, B. Random Features for Large-Scale Kernel Machines. In Proceedings of the Advances in Neural Information Processing Systems, Vancouver, BC, Canada, 3–6 December 2007; Platt, J., Koller, D., Singer, Y., Roweis, S., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2007; Volume 20..
28. Wilson, J.T.; Borovitskiy, V.; Terenin, A.; Mostowsky, P.; Deisenroth, M.P. Pathwise conditioning of Gaussian processes. *J. Mach. Learn. Res.* **2021**, *22*, 1–47.
29. Bradbury, J.; Frostig, R.; Hawkins, P.; Johnson, M.J.; Leary, C.; Maclaurin, D.; Necula, G.; Paszke, A.; VanderPlas, J.; Wanderman-Milne, S.; et al. JAX: Composable Transformations of Python+NumPy Programs. 2018. Available online: https://github.com/jax-ml/jax (accessed on 7 January 2025).
30. Metropolis, N.; Rosenbluth, A.W.; Rosenbluth, M.N.; Teller, A.H.; Teller, E. Equation of State Calculations by Fast Computing Machines. *J. Chem. Phys.* **1953**, *21*, 1087–1092. [CrossRef]
31. Girolami, M.; Calderhead, B. Riemann Manifold Langevin and Hamiltonian Monte Carlo Methods. *J. R. Stat. Soc. Ser. B Stat. Methodol.* **2011**, *73*, 123–214. [CrossRef]
32. Duane, S.; Kennedy, A.; Pendleton, B.J.; Roweth, D. Hybrid Monte Carlo. *Phys. Lett. B* **1987**, *195*, 216–222. [CrossRef]
33. Homan, M.D.; Gelman, A. The No-U-turn sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo. *J. Mach. Learn. Res.* **2014**, *15*, 1593–1623.
34. Zhang, R.; Li, C.; Zhang, J.; Chen, C.; Wilson, A.G. Cyclical Stochastic Gradient MCMC for Bayesian Deep Learning. In Proceedings of the International Conference on Learning Representations 2020, Virtual, 26 April–1 May 2020.
35. Cabezas, A.; Corenflos, A.; Lao, J.; Louf, R. BlackJAX: Composable Bayesian inference in JAX. *arXiv* **2024**, arXiv:2402.10797.
36. Van Rossum, G.; Drake, F.L., Jr. *Python Tutorial*; Centrum voor Wiskunde en Informatica: Amsterdam, The Netherlands, 1995.
37. Foreman-Mackey, D.; Yu, W.; Yadav, S.; Becker, M.R.; Caplar, N.; Huppenkothen, D.; Killestein, T.; Tronsgaard, R.; Rashid, T.; Schmerler, S. *dfm/tinygp: The Tiniest of Gaussian Process Libraries*; Zenodo: Geneva, Switzerlnad, 2024. [CrossRef]
38. Geyer, C.J. Practical Markov Chain Monte Carlo. *Stat. Sci.* **1992**, *7*, 473–483. [CrossRef]
39. Gelman, A.; Rubin, D.B. Inference from Iterative Simulation Using Multiple Sequences. *Stat. Sci.* **1992**, *7*, 457–472. [CrossRef]
40. Shahriari, B.; Bouchard-Cote, A.; Freitas, N. Unbounded Bayesian Optimization via Regularization. In Proceedings of the 19th International Conference on Artificial Intelligence and Statistics; Proceedings of Machine Learning Research, Cadiz, Spain, 9–11 May 2016; Gretton, A., Robert, C.C., Eds.; Volume 51, pp. 1168–1176.

41. Jamil, M.; Yang, X.S. A literature survey of benchmark functions for global optimisation problems. *arXiv* **2013**, arXiv:1308.4008. [CrossRef]
42. Surjanovic, S.; Bingham, D. Virtual Library of Simulation Experiments: Test Functions and Datasets. Available online: http://www.sfu.ca/~ssurjano (accessed on 18 November 2024).
43. Wang, Z.; Gehring, C.; Kohli, P.; Jegelka, S. Batched Large-scale Bayesian Optimization in High-dimensional Spaces. *arXiv* **2017**, arXiv:1706.01445.
44. Wang, Z.; Mohamed, S.; De Freitas, N. Adaptive Hamiltonian and Riemann manifold Monte Carlo samplers. In Proceedings of the 30th International Conference on Machine Learning, ICML'13, Atlanta, GA, USA, 16–21 June 2013; Volume 28, pp. III–1462–III–1470.
45. Zabinsky, Z.B.; Smith, R.L. Hit-and-Run Methods. In *Encyclopedia of Operations Research and Management Science*; Gass, S.I., Fu, M.C., Eds.; Springer US: Boston, MA, USA, 2013; pp. 721–729. [CrossRef]