Resource Efficient Bayesian Optimization

Namit Juneja, Varun Chandola, Jaroslaw Zola
Dept. of Computer Science and Engineering
University at Buffalo
USA
{namitjun,chandola,jzola}@buffalo.edu

Olga Wodo, Parth Desai
Dept. of Materials Design and Innovation
University at Buffalo
USA
{olgawodo,parthsam}@buffalo.edu

2024 IEEE 17th International Conference on Cloud Computing (CLOUD) | 979-8-3503-6853-6/24/$31.00 ©2024 IEEE | DOI: 10.1109/CLOUD62652.2024.00012

***Abstract*—We propose a resource-efficient Bayesian Optimization (BO) formulation that can provide the same convergence guarantees as traditional BO, while ensuring that the optimization makes efficient use of the available cloud or high-performance computing (HPC) resources. The paper is motivated by the fact that for many optimization problems that lend themselves well to BO, like hyper-parameter optimization for training large machine learning models, the single function evaluation cost depends on the model parameters as well as system parameters. The proposed Resource Efficient Bayesian Optimization (REBO) algorithm is a novel formulation that exploits this dependence and provides significant cost benefits for users who want to deploy BO on cloud and HPC resources that are characterized by availability of compute resources with varying costs and expected performance benefits. We demonstrate the effectiveness of REBO, in terms of convergence and resource-efficiency, on a variety of machine learning hyper-parameter optimization applications.**

***Index Terms*—Bayesian optimization, Resource-efficient optimization, Expected Improvement, Gaussian processes, active learning**

I. Introduction

Bayesian optimization (BO) [6] is a class of machine-learning based optimization methods that are well-suited for complex "black-box" objective functions of the form $f(\mathbf{x}) : \mathcal{X} \rightarrow \mathbb{R}$ over a set $\mathcal{X} \subset \mathbb{R}^d$, that lack the analytical form and gradients needed by first- and second-order derivative-based methods. A typical BO strategy is to approximate the true objective function using a surrogate function, represented using a Bayesian machine learning technique – Gaussian process regression (GPR) [5]. The GPR formulation allows BO to quantify the uncertainty in the surrogate at different candidate input points, and use the uncertainty to guide the search for next best candidate to sample for evaluation. BO has been shown to be effective for optimization over continuous domains of fewer than 20 dimensions ($d \leq 20$), by being *sample efficient*, i.e., taking fewer sample evaluations to converge than competing methods.

Ability to work with black box objective functions has made BO a highly versatile method in scenarios where a single evaluation of the target objective function can take several hours to evaluate. This includes applications such as hyper-parameter optimization for training large machine learning models [10], [21], [22], optimal parameter estimation for materials simulations [25], [27], drug design [17], etc. In each of these scenarios, the function evaluation is typically very costly, e.g, training a deep neural network model, running a computer simulation of a physical process, etc.

For many optimization problems, the evaluation cost varies across the search space, i.e., the evaluation is computationally more expensive for some values of $\mathbf{x}$ and cheaper for others [13]. For example, when training a deep neural network, the training cost depends on the value of the hyper-parameters, say, the number of layers of the network. Recent works have shown that by exploiting the heterogeneity in the computational cost profile, one can converge to the optimal solution while reducing the computational cost instead of the number of evaluations; being *cost-efficient* instead of *sample-efficient* [7], [13], [22], which is more desirable from the cost perspective.

However, above solutions that advocate cost-efficient BO, assume that the function evaluation is done on a single static compute environment. This is often not the case when the evaluation is done using a cloud or cluster computing platform. Such platforms typically offer different types of computing nodes with different compute characteristics (number of cores, availability of accelerators such as GPU(s), memory, etc.). The application that is used for the function evaluation might support parallel processing, and thus the user might also have the option of choosing the number of compute nodes for executing the application.

In this paper, we present a methodology to run BO on such systems where the user has the ability to choose different types of compute configurations at every iteration of BO, with an objective to converging to the solution while reducing the cost associated with using the system[1]. Clearly, simply choosing the cheapest or the most expensive configuration or a random configuration is not the best approach. For instance, while the cheapest configuration might have a lower per-hour cost, the evaluation might take longer and thus the actual dollar value for that evaluation might be higher. The application running the evaluation function itself could run faster or slower, for the

Partially supported by the National Science Foundation (Office of Advanced Cyberinfrastructure - OAC 1910539 and Division of Civil, Mechanical, and Manufacturing Innovation - CMMI 1906344. Computing facilities were provided by the University at Buffalo Center for Computational Research.

[1] By cost, we refer to dollar amount spent to run the evaluations. All commercial cloud providers as well as many high-performance computing (HPC) centers that offer cluster computing have per-hour pricing models that depends on the system configuration [2], [3], [18].

2159-6190/24/$31.00 ©2024 IEEE
DOI 10.1109/CLOUD62652.2024.00012

same $\mathbf{x}$, depending on the system configuration. For instance, a neural network training application might run faster if the compute node has a GPU or larger memory.

In general, we model the cost incurred for a single evaluation of $f(\mathbf{x})$ as:

$$c(\mathbf{x}, \boldsymbol{\theta}) = u(\boldsymbol{\theta}) \times t(\mathbf{x}, \boldsymbol{\theta}) \tag{1}$$

where the vector $\boldsymbol{\theta}$ represents the system configuration parameters, the function $u(\boldsymbol{\theta})$ models the per-unit-time price of acquiring the configuration $\boldsymbol{\theta}$, and the function $t(\mathbf{x}, \boldsymbol{\theta})$ models the time taken to run the function evaluation at $\mathbf{x}$ using the configuration $\boldsymbol{\theta}$. Note that while $u(\boldsymbol{\theta})$ (pricing model) is typically known, the time function $t(\mathbf{x}, \boldsymbol{\theta})$ is not known *a priori*.

Using the above cost formulation, we present a novel BO algorithm called *Resource Efficient Bayesian Optimization* or REBO, which identifies the optimal configuration ($\boldsymbol{\theta}$) to execute the next function evaluation at every step of the primary BO algorithm. REBO does not modify the primary BO algorithm (thus guarantees same convergence as BO[2]), but ensures that overall execution incurs lower cost. The time-cost function $t()$ is modeled using a second GPR model which iteratively admits more data after every BO evaluation and approximates the true but unknown function.

In this paper, we make the following contributions:

1) We advocate a resource-effective formulation of BO which offers a more practical approach to reducing running costs when deploying BO applications in cloud and HPC environments.
2) We propose REBO, which is a novel approach that can be used to deploy a BO application on a cloud system in the most cost-effective manner.
3) We demonstrate the effectiveness of REBO, using experiments for a variety of machine learning training applications, including neural network hyper-parameter tuning.

The rest of the paper is organized as follows. We provide an overview of BO and discuss relevant related works in Section II. The proposed REBO algorithm is presented in Section III. We provide an empirical evaluation and comparison with other related methods in Section IV and conclusions are provided in Section V.

II. Background and Related Work

A. Bayesian Optimization

Bayesian Optimization (BO) is a sequential method used for optimizing expensive-to-evaluate black-box functions. Given an objective function $f : \mathcal{X} \to \mathbb{R}$ our goal is to find $\mathbf{x}^*$ such that $f(\mathbf{x}^*) \leq f(\mathbf{x}), \forall \mathbf{x} \in \mathcal{X}$ where $\mathcal{X} \in \mathbb{R}^d$ and is referred to as the *search space*.

BO models the objective function f by iteratively training a *probabilistic surrogate model*. Typically this surrogate is built using a Gaussian Process (GP). Given n observations, $\{\mathbf{x}_i \in \mathcal{X}\}_{i=1}^n$, GP builds an n dimensional multivariate Gaussian Distribution $\mathcal{GP}_f(\mu(\mathbf{x}), k(\mathbf{x}, \mathbf{x}'))$ where $\mu : \mathcal{X} \to \mathbb{R}$ is the mean function and $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is the covariance function. The ability of a GP to estimate a function is greatly determined by what covariance function is used. A covariance function represents the prior knowledge we have about the objective function. A popular choice is the *squared exponential kernel*

$$k_{SE} = \sigma_s^2 \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2l^2}\right) \tag{2}$$

where variance σ_s^2, and length-scale l are kernel hyper-parameters that determine how the function values vary from their original mean and how smooth the function is respectively.

Once n evaluations of the objective function have been completed, $\{\mathbf{x}_i, y_i\}_{i=1}^n$ where $\mathbf{x}_i \in \mathcal{X}$ and $y_i = f(\mathbf{x}_i) + \epsilon_i$ and $\epsilon_i \sim \mathcal{N}(0, \sigma_\epsilon^2)$ is the input noise, the posterior distribution of the GP is given by $f(\mathbf{x}) \mid \{\mathbf{x}_{1:n}, y_{i:n}\} \sim \mathcal{N}(\mu_n(\mathbf{x}), \sigma_n^2(\mathbf{x}))$ where $\mu_n(\mathbf{x})$ is the mean and $\sigma_n^2(\mathbf{x})$ is the covariance at the current iteration and is defined as:

$$\mu_n(\mathbf{x}) = \mathbf{K}_*^T[\mathbf{K} + \sigma^2\mathbf{I}]^{-1}\mathbf{y} \tag{3}$$

$$\sigma_n^2(\mathbf{x}) = \mathbf{K}_{**}^T - \mathbf{K}_*^T[\mathbf{K} + \sigma^2\mathbf{I}]^{-1}\mathbf{K}_* \tag{4}$$

where y is an array of observed function evaluations $(y_i \ldots y_n)$, $\mathbf{K}$ is a covariance matrix between all observed points, $\mathbf{K}_*$ is the covariance between all observed points and new points, $\mathbf{K}_{**}$ is the covariance matrix between all new points, $\mathbf{I}$ is an identity matrix of dimensions same as that of $\mathbf{K}$ and σ_ϵ is the observed noise in the evaluation of the objective function.

An acquisition function, $\alpha : \mathcal{X} \to \mathbb{R}$ uses the mean μ_n and variance σ_n of the GP posterior to sample the next candidate points on which the objective function, f should be evaluated. The function's goal is to find $\mathbf{x}/ \in \mathcal{X}$ such that $\alpha(\mathbf{x}/) > \alpha(\mathbf{x}) \forall \mathbf{x} \in \mathcal{X}$. A popular choice for α is the *Expected Improvement*(EI) [14] function which is defined as:

$$\mathbf{x}_{n+1} = \arg\max_{\mathbf{x} \in \mathcal{X}}(\mathbb{E}(\max\{0, (\mu(x) - f(\tilde{\mathbf{x}})\} \mid \{\mathbf{x}_{1:n}, y_{1:n}\}))) \tag{5}$$

As the name suggests, EI computes the expected value of improvement over the current best evaluation over f at any point $\mathbf{x} \in \mathcal{X}$. Other well known acquisitions include *Probability of Improvement* (PI) [9], *Upper Confidence Bound* (UCB) [24] and *Entropy Search* (ES) [8]. Most modifications of BO that have been proposed in the past that make BO cost-efficient propose a modification to the EI acquisition function such that the cost of evaluation of the candidate points is also taken into consideration. We discuss these methods in detail in the following subsection.

B. Cost Efficient Bayesian Optimization

Many cost-efficient BO algorithms extend the traditional BO framework by incorporating evaluation costs. These methods

[2] This is not true of other cost-efficient methods [7], [13], [22] which modify the core search strategy of BO and do not provide similar guarantees about convergence.

develop EI based acquisition strategies that balance the trade-off between maximizing expected improvement and minimizing exploration cost. We explore three such popular acquisition function in this section.

One common approach to incorporate cost-efficiency into BO is to replace the EI acquisition function with the Expected Improvement per unit time (EIpu) acquisition function [23]:

$$EIpu(\mathbf{x}) = \frac{EI(\mathbf{x})}{m(\mathbf{x})} \tag{6}$$

where $m(\mathbf{x})$ is the predicted cost, typically modeled using a Warped GP fitted on the *log-cost* [23]. This approach favors points with high expected improvement and low predicted evaluation cost. However, [12] showed that EIpu fails to outperform EI in optimization problems where the optimum lies in a high-cost region. The acquisition function struggles to differentiate between high-cost regions with high expected improvement and low-cost regions with low expected improvement, as both yield similar EIpu values.

Eric et al. [12] proposed a cost-cooling modification to EIpu where the predicted cost used to normalize the EI is now exponentiated by a decaying function β.

$$EI - cool(\mathbf{x}) = \frac{EI(\mathbf{x})}{m(\mathbf{x})^{\beta}} \tag{7}$$

where β assumes the value of 1 at the beginning of the optimization and decays to 0 as the optimization progresses and the allocated cost budget slowly diminishes. In the beginning, EI-cool behaves similar to the original EIpu, favoring points with high expected improvement and low predicted cost, but as the optimization progresses, the predicted cost term becomes less influential eventually transforming the function into EI.

Guinet et al. [7] attempts to make BO cost-efficient by formulating the acquisition of new points as a bi-objective optimization problem. The optimal points are present on a pareto frontier where an optimal trade off between expected improvement and cost of evaluation is achieved. The Contextual Expected Improvement (CEI) acquisition function is defined as:

$$CEI_{\lambda}(\mathbf{x}) = \begin{cases} -c(\mathbf{x}) & EI(\mathbf{x}) \geq (1-\lambda) max_{\mathbf{z}\in\mathcal{X}}(EI(\mathbf{z})) \\ -\inf & otherwise \end{cases} \tag{8}$$

Our experimental results, presented in Table I, reveal that both cost-cooling and pareto-efficient strategies demonstrate a lack of flexibility across diverse search spaces. Moreover while these strategies improve cost-efficiency to some extent by improving the candidate acquisition process, they still operate under the assumption that the entire optimization is performed on a static system configuration. This assumption leaves a significant portion of the cost optimization potential untapped.

III. Proposed Resource-Efficient Bayesian Optimization Algorithm

Our work attempts to address the following problem: Given some objective function f, how can we use BO to optimize f in the minimum cost possible without compromising on the quality of optimization.

This problem is inherently challenging due to the intrinsic trade-off present in BO between selecting candidate points that facilitate faster convergence and those that incur the least evaluation cost. Previous solutions have attempted to resolve this dilemma by striking a balance between these two competing objectives.

However, a key observation is that these existing solutions operate under the assumption of system agnosticism, wherein each iteration of the BO is evaluated on the same system configuration, limiting the potential for cost savings.

In our work, we propose a novel framework, termed *Resource Efficient Bayesian Optimization*(REBO), which aims to drastically reduce the cost of optimization without sacrificing the quality of the optimization process. This is achieved by leveraging an auxiliary cost model, we call *Resource Cost Model* and the ability to dynamically adjust the system configuration across BO iterations.

REBO deviates from the traditional approach of striking a balance between selecting points that accelerate convergence and those that incur lower evaluation costs. Instead, it employs a greedy strategy that decouples the acquisition process into two distinct stages.

In the initial stage, the algorithm identifies a subset of points that offer the highest expected improvement, by maximizing the α_{EI} acquisition function. This stage is solely focused on exploiting regions within the search space that exhibit the greatest potential for improvement, effectively prioritizing the exploration of promising areas without considering the associated evaluation costs.

Subsequently, in the second stage, the algorithm shifts its focus to minimizing the evaluation costs. From the previously identified subset of points with the highest expected improvement, the algorithm selects the system configurations that would incur the least cost for evaluating the objective function. This resource-efficient selection process is driven by the Resource Cost Model (γ), which estimates the dollar cost associated with evaluating f, for each set of model and system parameters.

A. Resource Cost Model

The Resource Cost Model (RCM), denoted as γ, is a Bayesian model that predicts the probability distribution of the cost associated with evaluating the objective function f for a given set of model parameters ($\mathbf{x}$) and system parameters ($\boldsymbol{\theta}$). This model is defined as a product of the time cost model γ_t represented by a Gaussian Process Regression (GPR), which is iteratively trained as new points are evaluated by BO and a unit-time cost function γ_c which is known *a-priori* based on the compute service being used.

The RCM, $\gamma : \mathcal{X} \times \Omega \rightarrow \mathbb{R}$ yields the predicted mean cost, μ and uncertainty σ for a given $\mathbf{x}$ and $\boldsymbol{\theta}$.

The conditional probability distribution for the predicted cost, $\gamma(\boldsymbol{\theta} \mid \mathbf{x}')$, can be derived from the posterior distribution of γ. This conditional distribution is leveraged to determine the optimal system parameters, $\boldsymbol{\theta}'$, for a given set of model parameters $\mathbf{x}'$, such that the predicted cost of evaluating f at $\mathbf{x}'$ is minimal. To achieve this, we minimize the upper confidence bound of $\gamma(\boldsymbol{\theta} \mid \mathbf{x}')$.

$$\boldsymbol{\theta}' = \underset{\boldsymbol{\theta}\in\Omega}{\arg\min}\left(\mu(\boldsymbol{\theta} \mid \mathbf{x}') + \lambda\sigma(\boldsymbol{\theta} \mid \mathbf{x}')\right) \tag{9}$$

where λ is a tunable parameter that controls the trade-off between minimizing the predicted cost and accounting for the uncertainty associated with the cost prediction.

B. Resource Efficient BO

Let $f : \mathcal{X} \times \Omega \rightarrow \mathbb{R}^2$ be a function that maps model parameter space $\mathcal{X}$ and system parameter space Ω to evaluation cost and function output. We aim to find $\mathbf{x}^* \in \mathcal{X}$ and $\boldsymbol{\theta}^* \in \Omega$ such that $f(\mathbf{x}^*, \boldsymbol{\theta}^*) \leq f(\mathbf{x}, \boldsymbol{\theta}) \forall \mathbf{x} \in \mathcal{X}, \boldsymbol{\theta} \in \Omega$

REBO begins by initializing two models: the surrogate model which is a Gaussian Process, $\mathcal{GP}_f : \mathcal{X} \rightarrow \mathbb{R}$ and the Resource Cost Model, $\gamma : \mathcal{X} \times \Omega \rightarrow \mathbb{R}$. These models are initialized with a set of initial observations of the objective function f, typically obtained through a space-filling strategy, such as Latin Hypercube Sampling. This strategy ensures a diverse initial set of points, providing a suitable starting point for the optimization process. Subsequently, an EI based acquisition function, $\alpha_{EI} : \mathcal{X} \rightarrow \mathbb{R}$ determines the next set of points that f should be evaluated on. It is crucial to note that the points selected here are chosen solely based on maximizing the expected improvement, without considering the associated evaluation costs.

Following the selection of the new points by the α_{EI} acquisition function, a conditional probability distribution is constructed from γ. This distribution maps the estimated cost of evaluating f for the given set of model parameters determined by α_{EI} and all possible system parameters. By minimizing the upper confidence bounds of γ conditioned on the chosen model parameters, the set of system parameters is obtained, on which f can be evaluated such that it incurs the least cost. Once the set of model parameters and their corresponding set of system parameters have been acquired, they are used to evaluate the objective function f. With the newly acquired observation, both the $\mathcal{GP}_f$ and γ models are updated to incorporate the additional information. This update step involves re-estimating the GP hyperparameters and updating the posterior distributions of the two models.

The REBO algorithm then checks for convergence criteria, such as the allocated budget (e.g., number of function evaluations or computational resources) being exhausted or a satisfactory solution being found. If the convergence criteria is not met, REBO iterates back to the step where a new set of model parameters and system parameters are acquired using the updated surrogate model and RCM.

This iterative process continues until the convergence criteria are satisfied, hence leveraging surrogate model ($\mathcal{GP}_f$) and the the Resource Cost Model (γ) to navigate the optimization manifold in a resource-efficient manner while maintaining the quality of the optimization process. The key steps of this process are summarized in Algorithm 1.

It is noteworthy that if the Resource Cost Model (γ) and the associated two-step acquisition process are omitted, the REBO algorithm becomes equivalent to a traditional Bayesian Optimization algorithm.

IV. Experimental Validation

A. Experimental Setup

We evaluate the performance of REBO using a test bench consisting of both synthetic and real-world applications. The synthetic test bench includes three *2d* objective functions: Ackley function (defined over $[-32, 32]^2$), Rosenbrock function (defined over $[-5, 10]^2$), and Matyas function (defined over $[-10, 10]^2$).

The real-world applications in our study focus on three widely used Hyper Parameter Optimization (HPO) problems. First, we consider an Artificial Neural Network (ANN) algorithm, which we optimize by tuning three hyperparameters: number of hidden layers (defined over $[2, 10]$), number of nodes per hidden layer (defined over $[10, 100]$), and training batch size (defined over $[10, 100]$). Next, we optimize a Decision Tree algorithm, whose search space is defined by three hyperparameters: maximum depth (defined over $[1, 50]$), minimum number of samples required to split an internal node (defined over $[0.1, 0.9]$), and number of features to consider for the best split (defined over $[0.1, 0.9]$). Finally, we explore a Random Forest algorithm, which is also characterized by three hyperparameters: maximum depth (defined over $[1, 50]$), minimum number of samples required to split a node (defined over $[0.1, 0.9]$), and maximum number of leaf nodes (defined over $[1, 51]$).

Each HPO problem is tested on three different datasets: *income* [11], *bean* [1], and *bank* [15], resulting in 9 different combinations of HPO problems. The *income* dataset predicts whether a household's income exceeds $50,000 based on the 1994 US Census database. The *bean* dataset classifies beans into 7 different categories based on their features. The *bank* dataset predicts if a client will subscribe to a term deposit based on marketing campaigns of a Portuguese banking institution. These tests are also accompanied by the cost function mentioned earlier.

To enable resource-efficient optimization, REBO's search space also consists of system parameters that define the system configuration for each objective function evaluation. We consider a *3d* search space consisting of the number of nodes (defined over $[2, 32]$), the number of CPU cores (defined over $[2, 10]$), and the active memory per node (defined over $[4, 64]$).

To execute the optimization process, a dollar cost must be associated with each point in the search space that is evaluated. Generally, this cost would be obtained by calculating the time required to evaluate a point and the cost per unit time of the system configuration on which it was evaluated. However, in this paper, instead of measuring the actual cost, we use a simulated cost function. This cost function follows the

Algorithm 1 REBO: Resource Efficient Bayesian Optimization

```
Sample $n_0$ random points from $\mathcal{X}$: $\mathbf{x}^{(1:n_0)}$
Sample $n_0$ random points from $\Omega$: $\boldsymbol{\theta}^{(1:n_0)}$
Let the model and system parameters' combined vector be $X^{(1:n_0)} \leftarrow \{\mathbf{x}^{(1:n_0)} : \boldsymbol{\theta}^{(1:n_0)}\}$
Observe function evaluations, $y_f^{(1:n_0)}$ and evaluation costs, $y_c^{(1:n_0)}$ by evaluating $f$ at $X^{(1:n_0)}$
Organize the observations into $D_f^{(1:n_0)} \leftarrow \{(\mathbf{x}^{(1:n_0)}, y_f^{(1:n_0)})\}$ and $D_c^{(1:n_0)} \leftarrow \{(X^{(1:n_0)}, y_c^{(1:n_0)})\}$
$n \leftarrow n_0$
while optimization stopping condition not met do
    Update surrogate model, $\mathcal{GP}_f$ with $D_f^{(1:n)}$ to obtain $\mu_f(\mathbf{x})$ and $\sigma_f(\mathbf{x})$
    Update RCM, $\gamma$ with $D_c^{(1:n)}$
    Initialize acquisition function $\alpha_{EI}(\mathbf{x}, (\mu_f, \sigma_f))$
    $\mathbf{x}^{(n+1)} \leftarrow \arg\max_{\mathbf{x} \in \mathcal{X}} \alpha_{EI}(\mathbf{x})$ such that $\mathbf{x}^{(n+1)} \notin D_f^{1:n}$
    Compute $\gamma(\boldsymbol{\theta} \mid \mathbf{x}^{(n+1)})$ to obtain $\mu_c(\boldsymbol{\theta} \mid \mathbf{x}^{(n+1)})$ and $\sigma_c(\boldsymbol{\theta} \mid \mathbf{x}^{(n+1)})$
    $\boldsymbol{\theta}^{(n+1)} \leftarrow \arg\min_{\boldsymbol{\theta} \in \Omega} (\mu_c + \lambda\sigma_c)$
    $X^{(n+1)} \leftarrow \{\mathbf{x}^{(n+1)} : \boldsymbol{\theta}^{(n+1)}\}$
    Observe $y_f^{(n+1)}$ and $y_c^{(n+1)}$ by evaluating the objective function $f$ at $X^{(n+1)}$
    $D_f^{(1:n+1)} \leftarrow D_f^{(1:n)} \cup \{(\mathbf{x}^{(n+1)}, y_f^{(n+1)})\}$
    $D_c^{(1:n+1)} \leftarrow D_c^{(1:n)} \cup \{(X^{(n+1)}, y_c^{(n+1)})\}$
    $n \leftarrow n + 1$
end while
return $(\mathbf{x}^*, \boldsymbol{\theta}^*) \in D_f^{(1:n)}$ such that $f(\mathbf{x}^*, \boldsymbol{\theta}^*) \leq f(\mathbf{x}, \boldsymbol{\theta}) \forall \mathbf{x} \in \mathbf{x}^{(1:n)}, \boldsymbol{\theta} \in \boldsymbol{\theta}^{(1:n)}$
```

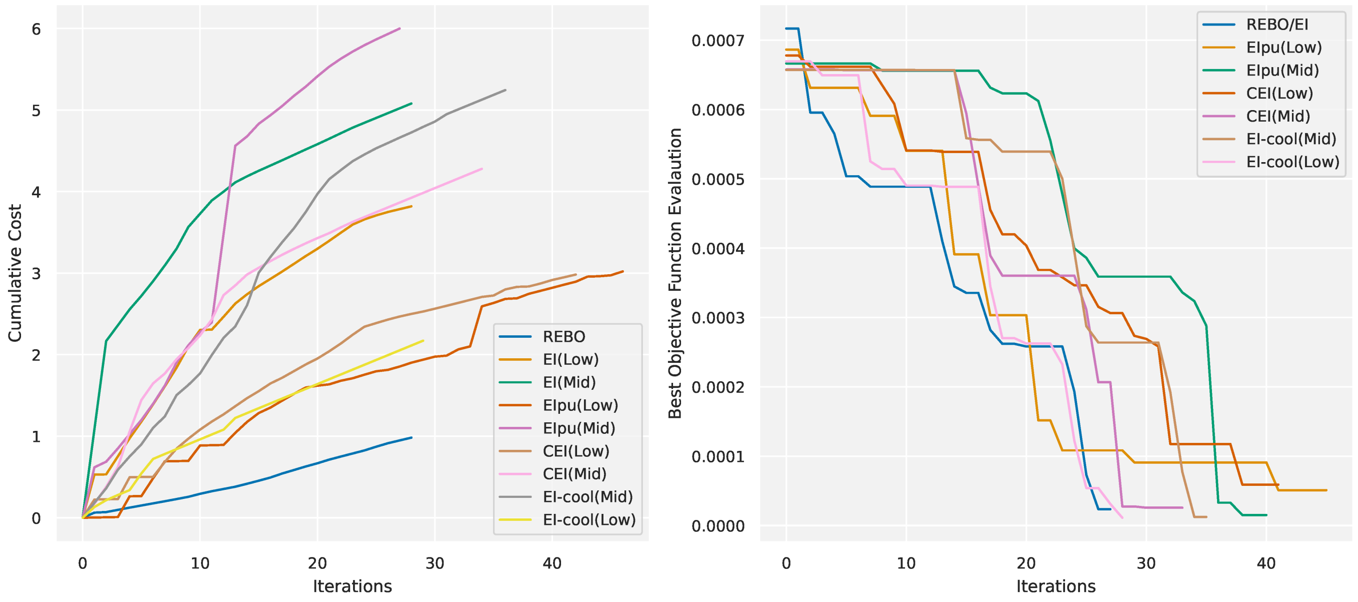

Fig. 1. Optimization comparison on a *2d* Ackley function. *Left:* Cumulative cost consumed by each algorithm. *Right:* Number of iterations it took for each algorithm to converge. REBO costs the least and is fastest to reach the optimum.

TABLE I
AVERAGE COST OF OPTIMIZATION ON SYNTHETIC AND REAL-WORLD TASKS

	EI	EI	EI	EIpu	EIpu	EIpu	CEI	CEI	CEI	EI-cool	EI-cool	EI-cool	REBO
	Low	Mid	High	Low	Mid	High	Low	Mid	High	Low	Mid	High	
Ackley	0.149	0.582	109.792	0.496	0.605	34.157	0.219	0.547	66.982	0.205	0.426	70.014	**0.084**
Rosen	0.125	0.257	11.619	0.157	0.279	9.170	0.135	0.255	10.889	0.143	0.208	15.119	**0.068**
Matyas	0.094	0.146	20.099	0.093	0.159	19.028	0.108	17.677	19.260	0.090	0.140	20.911	**0.062**
ANN *income*	0.083	0.088	0.737	0.076	0.437	3.879	0.308	0.924	8.767	0.097	0.764	76.379	**0.066**
ANN *bank*	0.229	0.690	7.983	0.259	3.826	5.874	0.542	0.363	7.289	0.121	0.827	4.024	**0.071**
ANN *bean*	0.059	0.240	0.374	0.073	0.287	8.742	0.426	0.827	2.837	0.115	0.349	12.939	**0.040**
Decision Tree *income*	0.030	0.068	2.953	0.106	0.110	0.699	0.055	0.166	7.705	0.039	0.120	8.203	**0.014**
Decision Tree *bank*	0.090	0.083	2.030	0.023	0.086	6.132	0.030	0.084	4.715	5.777	0.071	0.028	**0.024**
Decision Tree *bean*	0.050	0.094	2.358	0.051	0.079	6.586	0.042	0.132	5.097	0.052	0.122	3.692	**0.014**
Random Forest *income*	0.045	0.041	0.796	0.054	0.064	3.090	0.062	0.054	2.366	0.031	0.096	1.154	**0.008**
Random Forest *bank*	0.038	0.027	0.645	0.028	0.048	5.396	0.017	0.042	2.009	0.023	0.048	2.440	**0.006**
Random Forest *bean*	0.035	0.041	1.156	0.030	0.084	5.339	0.045	0.089	3.045	0.032	0.123	2.406	**0.007**

TABLE II
AVERAGE NUMBER OF ITERATIONS NEEDED TO OPTIMIZE ON SYNTHETIC AND REAL-WORLD TASKS

	EI	EI	EI	EIpu	EIpu	EIpu	CEI	CEI	CEI	EI-cool	EI-cool	EI-cool	REBO
	Low	Mid	High	Low	Mid	High	Low	Mid	High	Low	Mid	High	
Ackley	**35.60**	**35.60**	**35.60**	40.50	39.50	38.20	36.90	37.15	36.50	37.60	38.60	38.90	**35.60**
Rosen	**5.20**	**5.20**	**5.20**	5.77	5.71	5.05	5.63	5.38	6.06	5.47	5.75	5.65	**5.20**
Matyas	23.57	23.57	23.57	23.40	25.20	**22.00**	25.70	20.50	21.70	22.50	22.40	24.20	23.57
ANN *income*	**15.70**	**15.70**	**15.70**	19.80	21.80	16.90	15.90	18.87	19.10	18.70	19.10	19.70	**15.70**
ANN *bank*	**21.50**	**21.50**	**21.50**	29.54	27.60	31.50	25.80	26.30	24.90	27.90	23.30	32.60	**21.50**
ANN *bean*	18.30	18.30	18.30	17.60	19.70	20.30	19.30	**15.70**	22.87	21.54	21.80	22.20	18.30
Decision Tree *income*	**30.03**	**30.03**	**30.03**	32.65	33.00	34.63	33.94	31.30	34.30	33.40	35.07	35.80	**30.03**
Decision Tree *bank*	**19.19**	**19.19**	**19.19**	23.81	25.90	24.36	23.87	25.43	23.25	22.72	23.30	20.14	**19.19**
Decision Tree *bean*	**25.03**	**25.03**	**25.03**	30.50	27.30	26.30	30.50	26.70	28.47	27.95	26.96	27.88	**25.03**
Random Forest *income*	**19.93**	**19.93**	**19.93**	24.36	20.73	21.10	23.60	21.50	22.90	23.20	23.00	24.17	**19.93**
Random Forest *bank*	11.76	11.76	11.76	15.63	**11.37**	13.45	13.42	12.84	13.15	15.59	12.73	14.20	11.76
Random Forest *bean*	**18.83**	**18.83**	**18.83**	18.90	20.20	22.50	19.40	19.80	22.20	21.95	21.48	19.27	**18.83**

same structure as eqn: 1, consisting of two components: a time cost function and a unit cost function. The time cost function is defined by $(n_1 - f_1)^2 + (n_2 - f_2)^2 + n_3$, where n_1, n_2, and n_3 are the three system parameters, and f_1 and f_2 are linear combinations of the different model parameters. This formulation reflects the observation that different model parameters corresponds to a different set of system parameters that yield the optimum evaluation cost, which is consistent with findings from previous works [16], [20], [26]. The second component of the cost function is the per-unit-time cost function, which is modeled after the pricing of *AWS EC2 c6gn* compute instances [1]. By combining these two components, the cost function provides a realistic estimate of the total cost associated with evaluating the objective function under different system configurations and model parameters.

We conduct a comparative analysis of REBO against state-of-the-art baselines (EIpu, CEI and EI-cool) as well as standard Bayesian Optimization using Expected Improvement (EI). Since these baseline methods do not consider the system configuration, we conduct the optimization under three different static system settings: a *high* system configuration (highest per-unit-time price) with 32 nodes, 10 cores, and 64GB memory; a *mid* configuration (moderate per-unit-time price) with 16 nodes, 6 cores, and 32GB memory; and a *low* system configuration (lowest per-unit-time price) with 3 nodes, 1 core, and 2GB memory. We compare the performance of these algorithms with REBO, which employs a dynamically adjusting system configuration that adapts as it learns more about the underlying distribution of the objective function.

Our implementation is based on the boTorch [4] and scikit-learn [19] libraries. To ensure a fair comparison, all algorithms across each benchmark employ the same kernels: a Matérn kernel for the surrogate model Gaussian Process (GP) and a Radial Basis Function (RBF) kernel for the RCM GP. Furthermore, all algorithms start from an identical set of initial points. The hyperparameters are tuned by optimizing the maximum marginal log-likelihood. Each optimization is repeated 100 times, and the reported results represent the average performance across these runs.

B. Experimental Results

Our evaluation of REBO against the aforementioned algorithms is based on two key criteria. First is the total cost incurred in reaching the optimum. For the synthetic functions reaching the optimum signifies reaching their respective global minima/maxima and for HPO, it is defined as achieving a 10% improvement in the prediction accuracy. Table I presents the average cost of optimization over 100 replications, showing that REBO outperforms EI, EIpu, CEI, and EI-Cool by a significant margin.

The second criterion we use to evaluate is the quality of optimization, which is quantified by the number of iterations required to reach the optimum. Table II summarizes the average number of iterations needed for each model to reach the optimum condition, across 100 replications. Our findings indicate that REBO reaches the optimum condition in fewer iterations compared to the other methods when the optimum lies in a high-cost region, while exhibiting comparable performance when the optimum exists in a low-cost region. It is worth noting that the number of iterations required for REBO to reach the optimum is identical to that of EI, as REBO also employs the Expected Improvement (EI) acquisition function.

We visualize the improvement that REBO offers by examining an individual optimization run of the *2d* Ackley function using REBO, EI, EIpu, CEI, and EI-cool across *low*, and *mid* system configurations in Figure 1. We omit the *high* system configuration from the plot due to its significantly higher optimization cost, which makes it difficult to visualize on the same scale as the other configurations. In Figure 1, we make two key observations. First, REBO is able to optimize the objective function in the least cost when compared to other methods. This is attributed not only to the lesser number of points being sampled to reach the optima but also to REBO's ability to dynamically select a system configuration at each iteration that can evaluate the sampled point in the most cost-effective manner. This is evident from the cumulative cost plotted across each iteration. Second, we observe that REBO maintains the same sample efficiency as EI by being able to find the minima in the minimum number of iterations.

V. Conclusions

In the current era of machine learning, where model parameters range from a few hundred to billions, it has become increasingly evident that simply scaling computational resources is not the most cost-effective strategy for training the models. Choosing the right type of resources is equally crucial in optimizing the cost-efficiency of the training process, which includes iterating over a large number of model hyper-parameters. For very large machine learning models, a single node often lacks the power to train these models quickly. As a result, it is essential to distribute the training across multiple nodes to meet the computational demand. However, performance does not scale linearly with the number of nodes. In some cases, spreading processes across many servers is more effective, while in others, consolidating them on a single server yields better results. Cloud and HPC platforms enable such possibilities by offering a heterogeneous mix of computing resources with varying performance and cost profiles.

We argue that BO, which has emerged as an excellent optimization method for black-box functions, needs to be modified so that users can exploit the available resources on cloud and HPC platforms effectively. The resource-efficient formulation of the proposed REBO algorithm is a novel strategy and the empirical results show that REBO provides the same convergence guarantees as the original BO, while providing significant cost benefits.

References

[1] Dry Bean. UCI Machine Learning Repository, 2020. DOI: https://doi.org/10.24432/C50S4B.

[2] AWS EC2 on-demand pricing. https://aws.amazon.com/ec2/pricing/on-demand/.

[3] Microsoft azure linux virtual machines pricing. https://azure.microsoft.com/en-us/pricing/details/virtual-machines/linux/.

[4] M. Balandat, B. Karrer, D. R. Jiang, S. Daulton, B. Letham, A. G. Wilson, and E. Bakshy. BoTorch: A Framework for Efficient Monte-Carlo Bayesian Optimization. In *Advances in Neural Information Processing Systems 33*, 2020.

[5] G. P. for Machine Learning. *Carl Edward Rasmussen and Christopher K. I. Williams*. The MIT Press, 2006.

[6] R. Garnett. *Bayesian Optimization*. Cambridge University Press, 2023.

[7] G. Guinet, V. Perrone, and C. Archambeau. Pareto-efficient acquisition functions for cost-aware bayesian optimization. *ArXiv*, abs/2011.11456, 2020.

[8] J. M. Hernández-Lobato, M. W. Hoffman, and Z. Ghahramani. Predictive entropy search for efficient global optimization of black-box functions. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[9] D. R. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13(4):455–492, Dec 1998.

[10] A. Klein, S. Falkner, S. Bartels, P. Hennig, and F. Hutter. Fast bayesian optimization of machine learning hyperparameters on large datasets. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 54 of *Proceedings of Machine Learning Research*, pages 528–536. PMLR, 2017.

[11] R. Kohavi. Census Income. UCI Machine Learning Repository, 1996. DOI: https://doi.org/10.24432/C5GP7S.

[12] E. H. Lee, V. Perrone, C. Archambeau, and M. Seeger. Cost-aware bayesian optimization. *arXiv preprint arXiv:2003.10870*, 2020.

[13] E. H. Lee, V. Perrone, C. Archambeau, and M. W. Seeger. Cost-aware bayesian optimization. *ArXiv*, abs/2003.10870, 2020.

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

[15] R. P. Moro, S. and P. Cortez. Bank Marketing. UCI Machine Learning Repository, 2012. DOI: https://doi.org/10.24432/C5K306.

[16] L. Muttoni, G. Casale, F. Granata, and S. Zanero. Optimal number of nodes for computation in grid environments. In *12th Euromicro Conference on Parallel, Distributed and Network-Based Processing, 2004. Proceedings.*, pages 282–289, 2004.

[17] D. M. Negoescu, P. I. Frazier, and W. B. Powell. The knowledge-gradient algorithm for sequencing experiments in drug discovery. *INFORMS Journal on Computing*, 23(3):346–363, 2011.

[18] Ohio supercomputer center service costs. https://www.osc.edu/content/academic_fee_model_faq.

[19] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[20] Program scalability analysis. https://cis.temple.edu/~shi/wwwroot/shi/public_html/super96/.

[21] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.

[22] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In F. Pereira, C. Burges, L. Bottou, and K. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[23] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In F. Pereira, C. Burges, L. Bottou, and K. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[24] N. Srinivas, A. Krause, S. M. Kakade, and M. W. Seeger. Information-theoretic regret bounds for gaussian process optimization in the bandit setting. *IEEE Transactions on Information Theory*, 58(5):3250–3265, 2012.

[25] T. Ueno, T. D. Rhone, Z. Hou, T. Mizoguchi, and K. Tsuda. Combo: An efficient bayesian optimization library for materials science. *Materials Discovery*, 4:18–21, 2016.

[26] W. Wang, G. Chen, H. Chen, T. T. A. Dinh, J. Gao, B. C. Ooi, K.-L. Tan, S. Wang, and M. Zhang. Deep learning at scale and at ease. *ACM Trans. Multimedia Comput. Commun. Appl.*, 12(4s), nov 2016.

[27] Y. Zhang, D. W. Apley, and W. Chen. Bayesian optimization for materials design with mixed quantitative and qualitative variables. *Scientific Reports*, 10(1):4924, 2020.