Scalable Hyperparameter Optimization for LLM Fine-Tuning with BO and Partition Methods

N. Davouse[1] and E-G. Talbi[2]

[1] Univ. Lille, CNRS, Inria, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France
nathan.davouse@inria.fr

[2] Univ. Lille, CNRS, Inria, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France
el-ghazali.talbi@univ-lille.fr

Abstract. Large Language Models (LLMs) have demonstrated state-of-the-art performance across a variety of natural language processing (NLP) tasks. However, fine-tuning these models for domain-specific applications remains computationally expensive due to the vast hyperparameter space involved. In this paper, we propose two complementary approaches to address the hyperparameter optimization (HPO) problem for LLM fine-tuning: Bayesian Optimization (BO) and a Partition-based Optimization method. BO offers an efficient exploration of the hyperparameter space by balancing exploration and exploitation, minimizing the number of costly function evaluations. In parallel, the Partition-based method divides the search space into regions, enabling concurrent optimization across partitions.
We compare these approaches and explore their potential integration into a hybrid strategy, drawing on concepts from Bayesian Multiscale Optimistic Optimization (BamSOO). This study aims to harness the scability of partition based optimization combined to the efficience of Bayesian Optimization to deal with very expensive function in HPC environment.

1 Introduction

With the advent of the transformer architecture, coupled with significant advancements in the field of High Performance Computing (HPC), Large Language Model (LLM)s have demonstrated exceptional capabilities in language understanding and text generation, firmly establishing themselves as the State-of-the-art (SOTA) in Natural Language Processing (NLP). Models such as GPT-4 [1] and LLaMA 3 [2] have rapidly permeated both personal and professional domains, transforming workflows across academic, industrial, and everyday applications.

Despite their successes, the computational and data requirements for training LLMs from scratch make this approach impractical for most users. Fine-tuning has emerged as a critical paradigm, adapting pre-trained models to specialized tasks using smaller domain-specific datasets. While fine-tuning reduces costs compared to full training, it remains resource-intensive, particularly for large-scale models. To address this, parameter-efficient fine-tuning (PEFT) techniques [3], such as Low-Rank Adaptation (LoRA) [4], have become prevalent. LoRA and its derivatives (e.g., QLoRA [5], LoRA+ [6]) reduce computational overhead by focusing on the low-rank subspaces of weight matrices, enabling scalable fine-tuning without compromising performance.

Even with advancements in PEFT, a significant barrier persists: selecting optimal hyperparameters. Unlike model parameters, which are learned during training, hyperparameters must be manually specified and profoundly influence model performance. For LoRA-based fine-tuning, hyperparameters such as rank and scaling factors are critical[7], alongside classical parameters like learning rate and weight decay.

Hyper-Parameters Optimization (HPO) is a well-studied problem [8,9,10] in automated deep neural network design [11]. However, HPO for LLM fine-tuning presents unique challenges due to the prohibitive cost of evaluating the objective function. Each evaluation entails computationally expensive training or fine-tuning processes, often requiring hours to days on high-performance hardware. These constraints necessitate the adoption of highly efficient optimization techniques.

Bayesian Optimization (BO) [12] is a widely recognized approach for tackling expensive black-box optimization problems. Its ability to model the search space using a surrogate function allows it to minimize costly evaluations. However, BO sequential nature limits its parallelization capabilities. In contrast, Partition Based Optimization (PBO) methods [13,14], which divide the search space into subregions, enable parallel evaluations but typically require more function calls to converge. Hybrid methods that integrate the strengths of BO and PBO, such as Bayesian Multi-scale Optimistic Optimization (BaMSOO) [15], have shown promise in balancing exploration and exploitation.

In this work, we propose and evaluate two complementary approaches to address the HPO problem for LLM fine-tuning: Bayesian Optimization (BO) and Partition-Based Optimization (PBO). By systematically comparing their performance and exploring hybrid strategies, we aim to identify scalable and effective solutions for HPO of very expensive objective function.

2 Problem definition

Hyper-Parameters Optimization (HPO) of LLM Fine-Tuning is a very recent problem in literature, with a first review on article [16], but HPO for dnn is a known field with the automl community. For further reading, a taxonomy of autodnn can be found [11].

This problem can be classified as a black-box objective function optimization, i.e. the objective function cannot be formulated analytically and can't be derived. This characteristic constrain the sets of methods to be used directly. Following article [17] notation, this problem is also a Mixed Variables Optimization Problem, w.r.t. the set of values possible for one hyperparameter. The last key aspect of this optimization problem is the cost of evaluating a solution; it can take minutes to hours, restraining the number of possible evaluations.

Given a model m, a training dataset $\mathcal{D}_{train}$, a validation dataset $\mathcal{D}_{val}$ and a set of hyperparameters η, the black-box function can be expressed as $\mathcal{F}(\eta, m, \mathcal{D}_{train}, \mathcal{D}_{val})$. This function includes the training of the model, and also the evaluation using $\mathcal{D}_{val}$, and allow to link hyperparameters to training and to evaluation. In this work, the optimization is solely single objective, so the result of the function $\mathcal{F}$ is in $\mathbb{R}$.

Given this function, the optimization problem can be expressed as equation 1, where $\mathcal{H}$ is the search space of hyperparameters. The aim of this problem is to find one value being the maximum of the function $\mathcal{F}(.,.,.,.)$.

$$\eta^* \in \arg\max_{\eta \in \mathcal{H}} \mathcal{F}(\eta, m, \mathcal{D}_{train}, \mathcal{D}_{val}) \tag{1}$$

2.1 Search Space

The search space is defined with the choice of hyperparameters, theirs bounds, theirs types and even the scale of theirs steps. Well-defined search space is crucial for a correct application of HPO : if the space is too high-dimensional, the HPO algorithm will need too many shots to converge to a good solution, if it's too small, we are missing it's relevance.

The search space is composed of 5 hyperparameters, from classical training hyperparameters to LoRA specific ones. A detailed presentation of hyperparameters is just below, but one can look at table 1 for a summary.

- LoRA rank : with LoRA methods, the fine-tuning weights matrix $\Delta W \in \mathbb{R}^{n*p}$ is replaced by two matrices $A \in \mathbb{R}^{r*p}$ and $B \in \mathbb{R}^{n*r}$ with $\Delta W = B * A$. r is called the LoRA rank, and scale the reduction of the weights matrix. It's an integer, and it's value range from 1 to 512, following article [18] indication.
- LoRA scale (α) : when merging pre-trained weights W_0 and Lora fine-tuned weights $B * A$, the scale α is weighting the influence of fine-tuning, with $W = W_0 + \frac{\alpha}{r} * (B * A)$. It's a continuous value, from 1 to 100.
- Learning rate : the learning rate is a classical hyperparameter used in HPO, weighting the gradient of each weight when doing backpropagation. It is often tuned in a logarithmic scale, to manage effectively the exploration.
- Dropout probability : based on article [19], dropout is a method used to prevent over-fitting, by randomly fixing cells/layers to zeroes during one iteration. Being a probability, it's bounds by 0 and 1.

– Weight decay : weight decay is used to improve generalization capacity, as proved in article [20], by reducing the weights at each iterations by a small value, to force the model to use new inputs. Typically, the parameter for weight decay is set on a logarithmic scale between 10^{-3} and 10^{-1}.

Hyper-parameter	**Optimization range**		**Conversion**
	Lower Bound	**Upper Bound**	
Learning Rate	-10	-1	$f(x) = 10^x$
LoRA Rank	2	32	$f(x) = \text{round}(x)$
LoRA scale (α)	16	64	$f(x) = \text{round}(x)$
LoRA Dropout	0	0.5	$f(x) = x$
Weight Decay	-3	-1	$f(x) = 10^x$

Table 1: Summary of Hyperparameter Search Space

For the 2 integers variables (LoRA rank and LoRA scale), to adapt to continuous optimization algorithms, the relax and round methods will be applied. It mean that the integers constraints is relaxed when generating a solution, and is rounded when evaluating a solution. Others methods like computing with lower and upper discrete value can be used, but this one was kept for simplicity and computation costs. For the 2 variables with logarithmic scale (learning rate and weight decay), to explore with consistency the search space, the optimization algorithm will be bound between the exponential values of the bounds, and a logarithm will be applied when evaluating a solution.

2.2 Objective Function

Classicaly with DNN, the function to optimize might be the loss , sometimes normalized like Cross-entropy Loss or the accuracy. To avoid data leakage, the function is computed over a validation dataset, or even a testing dataset, different from the training one. For LLM, the datasets are a lot more diverse than for images classification, and the loss is biased to value the performance of a LLM, since it take all probabilities inside the function.

A second option to evaluate LLM fine-tuning is to use the accuracy over standard benchmarks, using multiple-questions choices to process an efficient evaluation of the performance of the LLM. Article like [21] explain that fine-tuned model have better generalization performance, so it's relevant to evaluate on a different dataset than the training, even if the layout of the inputs change. State-of-the-art example for fine-tune model are HELLASWAG[22] or MMLU[23,24]. In this papers, the function to maximize with the HPO is the accuracy on the Hellaswag datasets.

2.3 Related work

To locate this work on a global fields, this section aims to review relevant problematic and contributions close to this paper.

– AutoDNN : the global autoDNN fields has been thoroughly explored last years. Article [11] make a review and a taxonomy of this global fields. HPO applied to LLM fine-tuning differs from this especially by the cost of evaluation. Methods or specific problem of autodnn are now infeasible with LLM due to the costs. Specific methods are then needed.
– LLM applied to EA : when looking at interaction between LLM are EA, a lot of articles [25,26,27] are using LLM to enhance or popularize EA and optimization algorithms. This approach isn't relevant to our contribution, since our contribution focus on our optimization expertise.
– Prompt optimization : when linking optimization and LLM, some papers [28,29] aims to optimize the prompt, and work with the entire LLM as a blackbox. This approach have limited effect on LLM, since the performance are still cap by model training, and they replace the end users on chatbot deployment.
– HPO applied to inference HP : some hyperparameters like the temperature are only used when the LLM are generating tokens. Some are working of theses hyperparameters, to avoid training costs of working directly with the fine-tuning hyperparameters.

3 Methodology

In this work, two approaches are considered to performs hyperparameters optimization : Bayesian-Optimization based on Gaussian Process (BO-GP) and Partition based with algorithms like Simultaneous Optimistic Optimization[14]. To extract the best of these two approachs, an hybrid approach will also be presented.

Figure 1 describe the generic workflow of HPO, to link with following sections aiming to describe sections of the workflow.

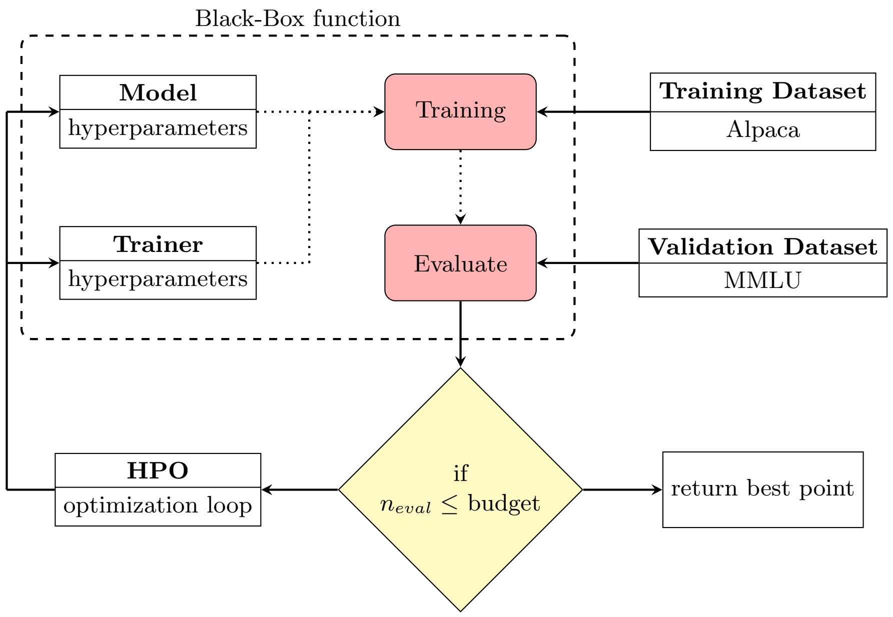


Fig. 1: Generic HPO workflow

The workflow is budget-based, meaning all algorithms will be compared with the same number of evaluations, i.e. the budget. The computing time of the algorithm is negligible in front of evaluation time, making the number of evaluation a relevant budget metric.

3.1 Fine-tune and evaluate LLM

HPO is a problem defined by a specific black-box function, and must adequate with the characteristic of this objective function. As said in 2, in this work, the function can be defined as an expensive, mixed-variables, noisy black-box function. This part aims to present this function, allowing to reproduce it.

The backbone model is LlaMa 3.2-3B [2], a model of the Llama 3 family of model. Llama 3 models were released by Meta AI in the second part of 2024 (from July to December), as open-source (topology and weights) decoder-only models achieving state-of-the-art performance. Among Llama 3 models, Llama 3.2 release propose lightweight models (from 1B to 3B weights, excluding vision models), enabling reproduction of the experiments.

The fine-tuning is perform using AdamW [30] optimizer, along LoRA methods to keep efficient fine-tuning. To achieve performance comparable with full fine-tuning, LoRA is used on keys, queries, values and attention head output weight, enabling the training of whole multi-head attention layers.

Then, the training data $\mathcal{D}_{train}$ is the *Alpaca* dataset, published after the *Standford Alpaca Project*. It's composed of 52k lines of IA generated and cleaned inputs, aiming to improve the behavior of a LLM. This dataset is widely used [5,31,32] for fine-tuning, guiding our choice for a standard configuration.

For evaluation, Hellaswag [22] dataset was used, with accuracy as the metrics to optimize with HPO. It's a 40k lines datasets release in 2019 as a challenge datasets, with a random performance of 25 % . To be precise, it's normalized accuracy that is used, to avoid favoring one choice over other. All models are also evaluated on MMLU [23] as a testing dataset, to observe over-fitting if present.

In terms of framework, all this part is done using LitGPT [33], a framework develop by Lightning AI team, based on Pytorch environment. This framework is thought for a command line interface utilization, and will be kept as it is in this work. Behind this framework, it's PyTorch [34] for training the model, Huggingface for loadings model and datasets, and lm_eval for managing evaluation.

3.2 Bayesian Optimization (BO) : Gaussian Process (GP) based optimization

Bayesian Optimization is often defined as a "sequential model-based approach to solving problem" [12]. A surrogate model is used to build a posterior considering a prior knowledge formed on known points. On this posterior, an acquisition function is compute to act as the surrogate function for the function to optimize. Many surrogate models can be used like regression tree [35] or Gaussian Process (GP) [36]. For further details about BO, one can read review [12].

On this work, a focus is done on GP for the BO surrogate. GP use the kernel trick to build a bayesian nonparametric regression model. It use a mean vector m_i and a covariance matrix $K_{i,j}$ to define the prior function :

$$\mathrm{f}|X \sim \mathcal{N}(m, K) \tag{2}$$

From the prior function and the data points $\mathcal{D}$, the GP build a posterior. On this posterior is build an acquisition function used as a surrogate for the objective function.

Algorithm 1 BO

```
Require: $\Omega$,$f$,$K_D$,$\mathcal{O}$,$f_{\mathrm{acq}}$,$n_{init}$,$n_{opt}$
 1: for $i = 1$ to $n_{init}$ do                         ▷ Initiate with Latin Hypercube Sampling
 2:     $\lambda' \leftarrow LHS(\Omega, \mathcal{D})$                          ▷ Sample one point
 3:     $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\lambda', f(\lambda'))\}$        ▷ Add solution and evaluation to set of data
 4: end for
 5:
 6: for $i = 1$ to $n_{opt}$ do                                        ▷ Optimization loop
 7:     $\mu_D, K_D \leftarrow \mathrm{Update}(K_D, \mathcal{D})$
 8:     $K_D \leftarrow \mathrm{Fit}(\mathrm{GP}(K_D), \mathcal{D})$
 9:     $\lambda' \leftarrow \mathrm{Optimize}(f_{\mathrm{acq}}(K_D), \mathcal{O})$                 ▷ Generate new point
10:     $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\lambda', f(\lambda'))\}$                  ▷ Evaluate new point
11: end for
12: return best of $\{(\lambda^*, f(\lambda^*)) \in \mathcal{D}\}$
```

Algorithm 1 offer an overview of the BO process. To ease the first build of the surrogate, it's crucial, as proven in article [37], to sample efficiently the search space. This sampling provides information for the Gaussian Process to estimation the function. Like article [38], Latin Hypercube Sampling (LHS) is used as a sampling method, for a defined budget called n_init.

After this preliminary phase, a second phase is done with on loop containing the update of the Gaussian Process, the optimization of the acquisition function to obtain a new points to evaluate. After the evaluation of the point, the point is added to the history $\mathcal{D}$ and so on. The loop end based on a budget n_{opt}.

For this algorithm, the first requirements is the search space, and the objective function already described in 2.1 and 2.2 respectively. On the GP part, we need to define a Kernel function $K_{\mathcal{D}}$, an acquisition function f_{acq} and an Inner Optimizer $\mathcal{O}$. The acquisition function is logEI, more reliable than EI, based on article [39]. The kernel and the inner optimizer are the standard implementation of Botorch, introduced in the next paragraph, with a radial basis function kernel and multi-start optimization method.

BoTorch [40] is a Bayesian Optimization library built on PyTorch, designed for efficient and scalable optimization of expensive black-box functions. Leveraging PyTorch's GPU acceleration and integration with GPyTorch [41] for Gaussian Processes, BoTorch enables advanced surrogate modeling and optimization. Botorch is used on this work for all tasks including GP, this part and section 3.4

3.3 Partition Based Optimization : Simultaneous Optimistic Optimization (SOO)

In global optimization, a lot of methods are based on the partition of the search space [13,42,14]. Theses approaches are mostly deterministic, and enhance intrinsic parallelization ability. For theses methods, the dimensionality of the problem is a key to choose the specific algorithm. With a dimensionality around 5, based on benchmarks at the end of article [43], the Simultaneous Optimistic Optimization (SOO) [14] algorithm seems a good way to start.

Algorithm 2 SOO

```
Require: $\Omega$,$f$,$K$,$n_{max}$
 1: $x_{0,0} \leftarrow center(\Omega)$
 2: $f_{0,0} \leftarrow f(x_{0,0})$
 3: $\mathcal{T}_1 = \{x_{0,0}, f_{0,0}, \Omega\}$
 4: $n \leftarrow 1$
 5: while $n < n_{max}$ do
 6:     $\nu_{max} \leftarrow -\infty$
 7:     for $h = 0$ to $depth(\mathcal{T}_n)$ do
 8:         $j \leftarrow \arg\max_{j \in \{j|(h,j) \in L_n\}} f(x_{h,j})$
 9:         if $f(x_{h,j}) > \nu_{max}$ then
10:             $\Omega_{h+1,j+1}, \dots, \Omega_{h+1,j+K} \leftarrow section(\Omega_{h,j}, K)$
11:             for $i = 1$ to$K$ do
12:                 $n \leftarrow n + 1$
13:                 $x_{h+1,j+i} \leftarrow center(\Omega_n)$
14:                 $f_{h+1,j+i} \leftarrow f(x_{h+1,j+i})$
15:                 $\mathcal{T}_n \leftarrow \{(x_{h+1,j+i}, f_{h+1,j+i}, \Omega_{n+1})\}$
16:             end for
17:             $\nu_{max} \leftarrow f_{h,j}$
18:         end if
19:     end for
20: end while
21: return best of $x_{h,j}, f(x_{h,j})$
```

SOO is a tree-based space partitioning method for black-box optimization, inspired by Monte Carlo Tree Search (MCTS) methods. SOO is called optimistic since it assume the existence of l such that $f(x^*) - f(x) \leq l(x, x^*)$ where x^* is the maximizer of x.The algorithm partition the space Ω by building a tree with smaller and smaller sub-space Ω_n. A node (h, j), the node number j of depth h, is scored at the center of his space.

An expanded node have K children, making the tree a K-nary tree. L_n is the *open list* of the tree, to avoid expanding the same nodes over and over. At each round, SOO expand a maximum of one node by depth, meaning that each round score a maximum of $depth * (K)$ solution, enhancing the parallel evaluation of the solution. Summary of SOO is present in algorithm 2

The original algorithm manage the end of the loop with the $h_{max}(n)$ function, limiting the depth of the tree search. To compare different algorithm, the stopping criterion here is n_{max}, the evaluation budget.

3.4 Hybrid approach : Bayesian Multi-scale Optimistic Optimization (BaMSOO)

Surrogate Model-Based Optimization (SMBO) algorithms harness the exploitation of the informations to define a cost-reduce function to optimize. This approach ensure exploitation but have several limitations, including the parallelization difficulties. On the other hand, Partition-based approach are massively parallel, but are computation costly in front of very expensive objective function. To overcome both limitations, hybrid methods, using surrogates and space partition, were developed.

In this work, we focus on BaMSOO, a SOO based algorithm (algorithm 3). Like SOO, BaMSOO performs a K-inary partitionning of the space, using the center of the partition to evaluate.

$$\begin{aligned} \mathcal{UCB}(x|\mathcal{D}_t) &= \mu(x|\mathcal{D}_t) + B_N * \sigma(x|\mathcal{D}_t) \\ \text{with } B_N &= \sqrt{2\log(\pi^2 N^2/6\eta)}, \eta \in (0,1) \end{aligned} \tag{3}$$

The difference lies primarily in the scoring $g(.)$ of the partitions (lines 25 to 30). In the face of an expensive objective function, BaMSOO leverages a GP surrogate to estimate the potential of a point, using the $\mathcal{UCB}$ as a measure of expected performance. Given a partition with center x and existing evaluations $\mathcal{D}_t$, the $\mathcal{UCB}$ of x, defined in Equation 3, is compared against the best evaluation so far, f^+. If the $\mathcal{UCB}$ is higher than f^+, the algorithm evaluates x directly using the objective function $f(.)$. Otherwise, the partition is scored using the LCB of x, reflecting the lower bound of potential improvement.

Algorithm 3 BamSOO

```
Require:
 1: $\Omega$: Continuous search space
 2: $f$: Objective function                              ▷ train and validate the model
 3: $K$ : number of section of the space
 4: $n_{max}$ : budget of evaluation
 5: $K_D$ : Kernel function
 6:
 7: $x_{0,0} \leftarrow center(\Omega)$
 8: $g_{0,0} \leftarrow f(x_{0,0})$
 9: $\mathcal{T}_1 = \{x_{0,0}, g_{0,0}, \Omega\}$                      ▷ Initiate the tree
10: $f^+ \leftarrow g_{0,0}$
11: $n \leftarrow 1$                                              ▷ nodes index
12: $t \leftarrow 1$                                          ▷ evaluation index
13: $\mathcal{D}_1 = \{x_{0,0}, g_{0,0}\}$                         ▷ list of evaluated points
14:
15: while $t < n_{max}$ do
16:     $\nu_{max} \leftarrow -\infty$
17:     for $h = 0$ to $depth(\mathcal{T}_n)$ do
18:         $j \leftarrow \arg\max_{j \in \{j|(h,j) \in L_n\}} g_{h,j}$
19:         if $g_{h,j} > \nu_{max}$ then
20:             $\Omega_{h+1,j+1}, \ldots, \Omega_{h+1,j+K} \leftarrow section(\Omega_{h,j}, K)$
21:             for $i = 1$ to $K$ do
22:                 $\mu, \sigma \leftarrow GP(\mathcal{D}_t, K_D)$
23:                 $N \leftarrow N + 1$
24:                 $x_{h+1,j+i} \leftarrow center(\Omega_n)$
25:                 if $\mathcal{UCB}(x_{h+1,j+i}, \mu, \sigma) \geq f^+$ then
26:                     $g_{h+1,j+i} \leftarrow f(x_{h+1,j+i})$
27:                     $t \leftarrow t + 1$
28:                 else
29:                     $g_{h+1,j+i} \leftarrow \mathcal{LCB}(x_{h+1,j+i}, \mu, \sigma)$
30:                 end if
31:                 if $g_{h+1,j+i} > f^+$ then
32:                     $f^+ \leftarrow g_{h+1,j+i}$
33:                 end if
34:                 $n \leftarrow n + 1$
35:                 $\mathcal{T}_n \leftarrow \{(x_{h+1,j+i}, f_{h+1,j+i}, \Omega_{h+1,j+i})\}$
36:             end for
37:             $\nu_{max} \leftarrow g_{h,j}$
38:         end if
39:     end for
40: end while
41: return best of $x_{h,j}, g(x_{h,j})$
```

To sum up, this algorithm efficiently navigates the search space by leveraging partition-based exploration and utilizing the gathered information to prioritize regions with higher potential. By balancing exploration of untested partitions and exploitation of known promising areas, BaMSOO optimally allocates the evaluation budget, ensuring that computational resources are directed toward regions most likely to contain optimal solutions. This hybrid approach significantly mitigates the limitations of traditional methods, combining the global scalability of partitioning with the precision of GP-based surrogate modeling.

For the implementation of the GP components, including the calculation of LCB and $\mathcal{UCB}$ scores, the BoTorch library was employed. This choice ensures computational efficiency and robustness, as BoTorch provides a modular framework for Bayesian optimization and GP modeling, seamlessly integrating with the partition-based structure of BamSOO. By adhering to the methodology outlined in section 3.2, the framework ensures consistency in surrogate modeling and acquisition function computation, further enhancing the effectiveness of the algorithm in high-dimensional, continuous search spaces.

4 Computation Experiments

The experiments are done with Grid5000 [44], " a large-scale and flexible testbed for experiment-driven research in all areas of computer science, with a focus on parallel and distributed computing including Cloud, HPC and Big Data and AI." In specific, the *chuc* cluster, in the Lille center, composed of nodes of 4 GPU A100 with 40G of VRAM was used.

Apart from aforementioned hyperparameters (learning rate, LoRA rank ...) and configuration (weights to apply LoRA) , along with the number of epochs, all arguments of litgpt CLI is used with default value. The number of epochs will be fixed with a first experiment to determine the sufficient number to be representative of a full training. For next sections, a difference will be made between variables (value inside optimization range), and hyperparameters (value used by the training function) to clarify the reading of the value.

Using this configuration, one epoch of fine-tuning is taking XX hours/minutes. For the evaluation, it's taking between X and XX minutes, depending on the dataset. Based on previous articles, and evaluations durations, the total evaluation budget for experiments is 50 evaluations by each algorithms, including a sampling budget of 10 for Bayesian Optimization.

Datasets	Lower	Upper
Hellaswag		69.8
MMLU		63.4

Table 2: Bounds on accuracy for validation and testing dataset

In face of such experiments, it's interesting to look at bounds of the metric, to compare the results. The lower bounds will be an experiments using solely LHS to pick the point, with the same number of evaluation than algorithms. LHS being intrinsicly parallel, if evaluated algorithms don't achieve better performance than a solely explorating one, their benefit isn't relevant. For the higher bound, I will take the higher value in the model card[3], achieved using advanced fine-tuning methods like Supervised Fine-Tuning (SFT), Rejection Sampling (RS), and Direct Preference Optimization (DPO). Values are inside table 2.

The implementation of the previous algorithm, the objective function and next experiments are all stored in github, following this link : https://github.com/Kiwy3/Scalable_HPO_LLM.

4.1 Determining sufficient number of epochs

Experiments from article [18] highlight that it's sufficient to train with two epochs to make difference between good and bad solutions. For this first experiment, the number of epochs is ranged from 1 to 4. On each epochs values, 5 solutions will be picked and then evaluated : the center of

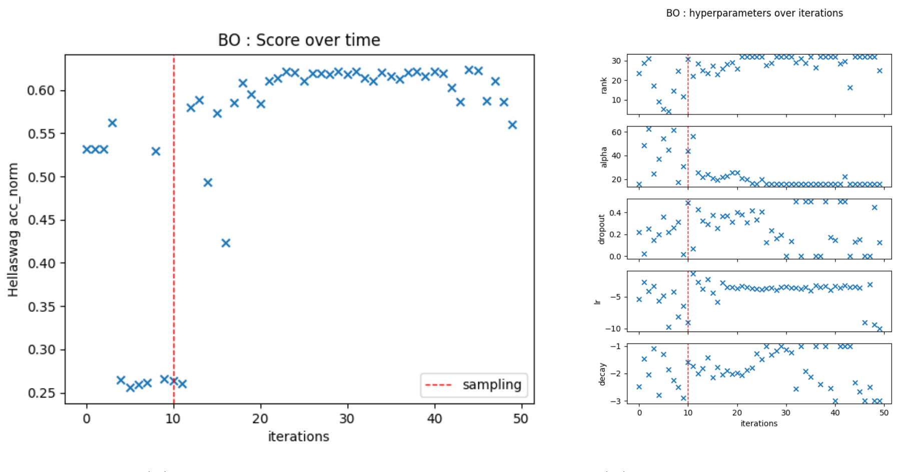
[3] https://huggingface.co/meta-llama/Llama-3.2-3B

the search space and 4 randoms picks. This way, with fixed random seed for training, we can look the increase between each epochs, to determine if it's worth to undergo another epochs for other experiments.

To conclude this section, the number of epochs of the next sections is fixed to X.

4.2 BO experiment

Figure 2 depicts the performance of Bayesian Optimization (BO) over 50 iterations, measured in terms of the normalized Hellaswag accuracy (acc_norm). This visualization highlights the evolution of the optimization process as it transitions from sampling to the exploitation, and ultimately converges towards high-performing solutions.



(a) Score over time (b) Variable values over time

Fig. 2: Experiment using BO algorithm

During the sampling phase, as shown by figure 2a the best score achieved is at 56.27% of normalized accuracy. This phase is interesting at is show us that there is a lot of area with good solution without exploitation. After this phase, there is around 15 iterations to converge to a stable phase, around 62% of accuracy. After 40 iterations, the algorithm renew with a phase of exploration, to look if it's possible to achieve best results in others configuration, resulting in it's decrease in performance.

If we briefly look at figure 2b, it allow to look at well-chosen optimization range, or if it can be useful to enlarge it. For example, when looking at LoRA rank, it's keeping it's value at the top of the range, implying that it would go up if possible.

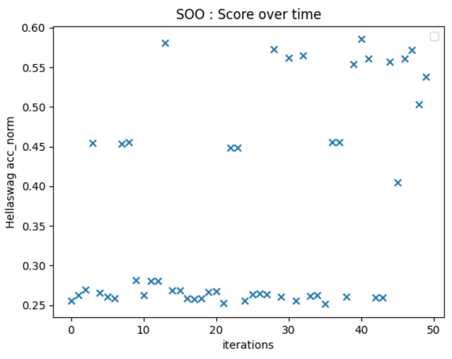
COMPARE WITH BOUNDS

To summarize, this experiment demonstrate the effectiveness of Bayesian Optimization in efficiently explore the search space. The optimization process strikes a balance between exploration and exploitation, achieving convergence within a limited number of iterations.

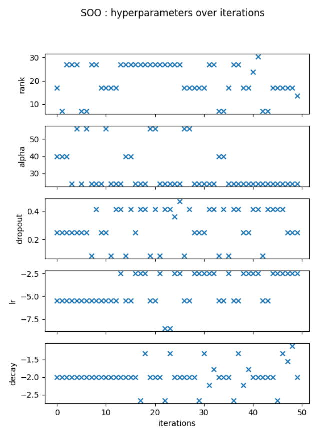
4.3 SOO experiment

Figure **??** depicts the performance of Simultaneous Optimistic Optimization (SOO) over 50 iterations, measured in terms of the normalized Hellaswag accuracy (acc_norm). This visualization highlights the progression of the optimization process, showing limited improvement until later iterations, when a significant rise in performance is observed.

In the initial phase, spanning roughly the first 20 iterations, the accuracy remains relatively low and consistent, fluctuating around 0.25 to 0.3. This indicates that the SOO algorithm invests heavily in exploration during the early stage, sampling diverse regions of the parameter space but failing to identify high-performing regions.



(a) Score over time



(b) Variables over time

Fig. 3: Experiment using SOO algorithm

Between iterations 20 and 40, a gradual improvement is observed as the algorithm transitions to exploitation. In this phase, SOO focuses on refining potential solutions, leading to a slight increase in accuracy, although the improvement remains modest compared to Bayesian Optimization.

In the final stage, after iteration 40, a significant rise in performance is observed, with accuracy values increasing sharply and converging around 0.6. This stabilization phase indicates that SOO successfully identifies high-quality parameter configurations, albeit at a slower pace compared to BO.

To summarize, Figure **??** demonstrates that SOO is capable of finding high-performing solutions but requires a longer exploration phase and slower convergence compared to Bayesian Optimization. Despite its delayed success, the eventual convergence highlights the potential of SOO for hyperparameter tuning tasks.

4.4 BaMSOO experiment

Figure 4 depicts the performance of Bamsoo over 50 iterations, measured in terms of the normalized Hellaswag accuracy (acc_norm). This visualization highlights a hybrid approach that balances the rapid improvement seen in Bayesian Optimization (BO) and the thorough exploration typical of Simultaneous Optimistic Optimization (SOO).

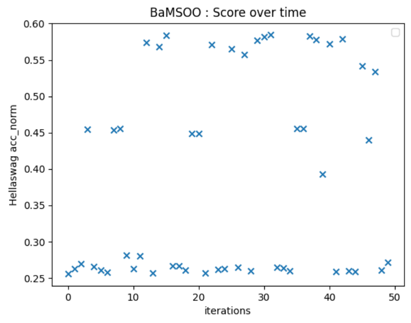


Fig. 4: Score over time with BaMSOO

In the initial phase, spanning the first 10 iterations, Bamsoo demonstrates an exploratory behavior similar to SOO. Accuracy values during this phase fluctuate around 0.25 to 0.3, indicating that the algorithm is sampling widely across the parameter space to gather foundational knowledge about the landscape.

Between iterations 10 and 30, the algorithm transitions to a phase of steady improvement, combining elements of exploration and exploitation. This phase is characterized by a gradual rise in accuracy, suggesting that Bamsoo refines its focus on promising regions of the parameter space while still allowing room for broader searches to avoid premature convergence.

In the final stage, after iteration 30, Bamsoo exhibits a stabilization phase with accuracy values consistently converging around 0.55 to 0.6. This indicates that the algorithm successfully identifies high-quality parameter configurations while maintaining the flexibility to avoid getting trapped in local optima.

To summarize, Figure 4 illustrates the hybrid nature of Bamsoo, which successfully balances exploration and exploitation across iterations. This behavior enables it to achieve competitive performance relative to BO and SOO, making Bamsoo a compelling alternative for optimization tasks requiring adaptability and robustness.

5 Conclusion

This study evaluated the performance of three optimization algorithms—Bayesian Optimization (BO), Simultaneous Optimistic Optimization (SOO), and Bamsoo—in hyperparameter tuning. BO showcased remarkable efficiency by balancing exploration and exploitation, achieving rapid convergence to high-performing solutions within minimal iterations. Its structured approach and smooth transition phases make it highly suitable for tasks requiring fast and accurate optimization.

SOO, on the other hand, exhibited a slower convergence trajectory, with a prolonged exploration phase followed by a delayed but notable improvement in performance. Although SOO's eventual performance aligned with BO, its longer stabilization period and higher iteration count highlight its computational inefficiency in time-sensitive scenarios.

Bamsoo, a hypothetical hybrid algorithm blending features of BO and SOO, demonstrated competitive performance by leveraging the strengths of both approaches. It balanced exploration and exploitation with adaptive flexibility, achieving steady improvement throughout its iterations. While not as swift as BO in convergence speed, Bamsoo surpassed SOO in terms of computational efficiency and reached high accuracy levels with fewer iterations. This makes Bamsoo a promising alternative for optimization tasks demanding both robustness and adaptability.

In conclusion, the findings highlight the dominance of BO in scenarios requiring rapid convergence and reliability. SOO's extended exploration capability positions it as a useful tool in contexts where thorough search is prioritized. Bamsoo's balanced performance suggests that hybrid strategies could serve as an optimal middle ground, paving the way for future research into hybrid optimization algorithms tailored to specific application needs.

Acknowledgment

This work was partially supported by the EXAMA (Methods and Algorithms at Exascale) project under grant ANR-22-EXNU-0002.

Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see https://www.grid5000.fr).

References

1. OpenAI, Achiam, J., Adler, S. et al.: GPT-4 Technical Report (2024) arXiv:2303.08774 [cs].
2. Grattafiori, A., Dubey, A., Jauhri, A. et al.: The Llama 3 Herd of Models (2024) arXiv:2407.21783 [cs].
3. Han, Z., Gao, C., Liu, J. et al.: Parameter-Efficient Fine-Tuning for Large Models: A Comprehensive Survey (2024) arXiv:2403.14608 [cs].
4. Hu, E.J., Shen, Y., Wallis, P. et al.: LoRA: Low-Rank Adaptation of Large Language Models (2021) arXiv:2106.09685.
5. Dettmers, T., Pagnoni, A., Holtzman, A. et al.: QLoRA: Efficient Finetuning of Quantized LLMs. Advances in Neural Information Processing Systems **36** (2023) 10088–10115
6. Hayou, S., Ghosh, N., : LoRA+: Efficient Low Rank Adaptation of Large Models (2024) arXiv:2402.12354 [cs].
7. Valipour, M., Rezagholizadeh, M., Kobyzev, I. et al.: DyLoRA: Parameter Efficient Tuning of Pre-trained Models using Dynamic Search-Free Low-Rank Adaptation (2023) arXiv:2210.07558 [cs].
8. Bischl, B., Binder, M., Lang, M. et al.: Hyperparameter Optimization: Foundations, Algorithms, Best Practices and Open Challenges (2021) arXiv:2107.05847 [stat].
9. Bergstra, J., Bardenet, R., Bengio, Y. et al.: Algorithms for Hyper-Parameter Optimization. Volume 24., Neural Information Processing Systems Foundation (2011)
10. Feurer, M., : Hyperparameter Optimization. In Hutter, F., Kotthoff, L., , eds.: Automated Machine Learning: Methods, Systems, Challenges. Springer International Publishing, Cham (2019) 3–33
11. Talbi, E.G.: Automated Design of Deep Neural Networks: A Survey and Unified Taxonomy. ACM Comput. Surv. **54** (2021) 34:1–34:37
12. Shahriari, B., Swersky, K., Wang, Z. et al.: Taking the Human Out of the Loop: A Review of Bayesian Optimization. Proceedings of the IEEE **104** (2016) 148–175 Conference Name: Proceedings of the IEEE.
13. Nakib, A., Ouchraa, S., Shvai, N. et al.: Deterministic metaheuristic based on fractal decomposition for large-scale optimization. Applied Soft Computing **61** (2017) 468–485
14. Munos, R.: Optimistic Optimization of a Deterministic Function without the Knowledge of its Smoothness. In: Advances in Neural Information Processing Systems. Volume 24., Curran Associates, Inc. (2011)
15. Wang, Z., Shakibi, B., Jin, L. et al.: Bayesian Multi-Scale Optimistic Optimization (2014) arXiv:1402.7005 [stat].
16. Liu, X., : An Empirical Study on Hyperparameter Optimization for Fine-Tuning Pre-trained Language Models (2021) arXiv:2106.09204 [cs].
17. Talbi, E.G.: Metaheuristics for variable-size mixed optimization problems: A unified taxonomy and survey. Swarm and Evolutionary Computation **89** (2024) 101642
18. Tribes, C., Benarroch-Lelong, S., Lu, P. et al.: Hyperparameter Optimization for Large Language Model Instruction-Tuning (2024) arXiv:2312.00949.
19. Srivastava, N., Hinton, G., Krizhevsky, A. et al.: (Dropout: A Simple Way to Prevent Neural Networks from Overfitting)
20. Krogh, A., : A Simple Weight Decay Can Improve Generalization. In: Advances in Neural Information Processing Systems. Volume 4., Morgan-Kaufmann (1991)
21. Wei, J., Bosma, M., Zhao, V.Y. et al.: Finetuned Language Models Are Zero-Shot Learners (2022) arXiv:2109.01652 [cs].
22. Zellers, R., Holtzman, A., Bisk, Y. et al.: HellaSwag: Can a Machine Really Finish Your Sentence? (2019) arXiv:1905.07830 [cs].
23. Hendrycks, D., Burns, C., Basart, S. et al.: Measuring massive multitask language understanding. Proceedings of the International Conference on Learning Representations (ICLR) (2021)
24. Hendrycks, D., Burns, C., Basart, S. et al.: Aligning ai with shared human values. Proceedings of the International Conference on Learning Representations (ICLR) (2021)
25. Liu, S., Chen, C., Qu, X. et al.: Large Language Models as Evolutionary Optimizers. In: 2024 IEEE Congress on Evolutionary Computation (CEC). (2024) 1–8

26. Cai, J., Xu, J., Li, J. et al.: Exploring the Improvement of Evolutionary Computation via Large Language Models. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion. GECCO '24 Companion, New York, NY, USA, Association for Computing Machinery (2024) 83–84
27. Brahmachary, S., Joshi, S.M., Panda, A. et al.: Large Language Model-Based Evolutionary Optimizer: Reasoning with elitism (2024) arXiv:2403.02054 [cs].
28. Diao, S., Huang, Z., Xu, R. et al.: Black-box Prompt Learning for Pre-trained Language Models (2023) arXiv:2201.08531 [cs].
29. Xu, H., Chen, Y., Du, Y. et al.: GPS: Genetic Prompt Search for Efficient Few-shot Learning (2022) arXiv:2210.17041 [cs].
30. Loshchilov, I., : Decoupled Weight Decay Regularization (2019) arXiv:1711.05101 [cs].
31. Chung, H.W., Hou, L., Longpre, S. et al.: Scaling Instruction-Finetuned Language Models. Journal of Machine Learning Research **25** (2024) 1–53
32. Zhou, C., Liu, P., Xu, P. et al.: LIMA: Less Is More for Alignment. Advances in Neural Information Processing Systems **36** (2023) 55006–55021
33. The Lightning AI team: LitGPT (2023) original-date: 2023-05-04T17:46:11Z.
34. Ansel, J., Yang, E., He, H. et al.: PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation (2024) Publication Title: 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24) original-date: 2016-08-13T05:26:41Z.
35. Ammari, B.L., Johnson, E.S., Stinchfield, G. et al.: Linear model decision trees as surrogates in optimization of engineering applications. Computers & Chemical Engineering **178** (2023) 108347
36. Rajaram, D., Puranik, T.G., Ashwin Renganathan, S. et al.: Empirical Assessment of Deep Gaussian Process Surrogate Models for Engineering Problems. Journal of Aircraft **58** (2021) 182–196 Publisher: American Institute of Aeronautics and Astronautics _eprint: https://doi.org/10.2514/1.C036026.
37. Wilson, J., Borovitskiy, V., Terenin, A. et al.: Efficiently sampling functions from Gaussian process posteriors. In: Proceedings of the 37th International Conference on Machine Learning, PMLR (2020) 10292–10302 ISSN: 2640-3498.
38. Borisut, P., : Adaptive Latin Hypercube Sampling for a Surrogate-Based Optimization with Artificial Neural Network. Processes **11** (2023) 3232 Number: 11 Publisher: Multidisciplinary Digital Publishing Institute.
39. Ament, S., Daulton, S., Eriksson, D. et al.: Unexpected Improvements to Expected Improvement for Bayesian Optimization (2024) arXiv:2310.20708 [cs].
40. Balandat, M., Karrer, B., Jiang, D.R. et al.: BoTorch: A Framework for Efficient Monte-Carlo Bayesian Optimization (2020) arXiv:1910.06403 [cs].
41. Gardner, J.R., Pleiss, G., Bindel, D. et al.: GPyTorch: Blackbox Matrix-Matrix Gaussian Process Inference with GPU Acceleration (2021) arXiv:1809.11165 [cs].
42. Jones, D.R., Perttunen, C.D., : Lipschitzian optimization without the Lipschitz constant. Journal of Optimization Theory and Applications **79** (1993) 157–181
43. Firmin, T., : A fractal-based decomposition framework for continuous optimization. (2022)
44. Balouek, D., Carpen-Amarie, A., Charrier, G. et al.: Adding Virtualization Capabilities to Grid'5000. report, INRIA (2012) Pages: 18.