Bayesian and Partition-Based Optimization for Hyperparameter Optimization of LLM Fine-Tuning

N. Davouse and E-G. Talbi

Univ. Lille, CNRS, Inria, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France
nathan.davouse@inria.fr and el-ghazali.talbi@univ-lille.fr

Abstract. Large Language Models (LLMs) have revolutionized natural language processing (NLP) by achieving state-of-the-art performance across diverse tasks. However, fine-tuning these models for domain-specific applications is significantly constrained by the computational costs associated with their training. In this paper, we propose two complementary approaches to address the Hyperparameter Optimization (HPO) challenge in LLM fine-tuning: Bayesian Optimization based on Gaussian Process (BO-GP) and Partition-Based Optimization (PBO). On the one hand, BO efficiently exploits historical knowledge to achieve optimal results within a limited number of evaluations, but its inherently sequential nature poses scalability challenges. On the other hand, PBO enables massive parallelization, making it more scalable but requiring significantly more evaluations to converge. To leverage their complementary strengths for optimizing expensive objective functions, we investigate these methods and propose a hybrid BO-PBO algorithm. This work represents a foundational step toward harnessing the potential of parallel Bayesian Optimization-based algorithms for solving expensive optimization problems in exascale computing environments.

1 Introduction

With the advent of transformer architecture[1], coupled with significant advancements in the field of High Performance Computing (HPC), Large Language Model (LLM)s have demonstrated exceptional abilities 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 [2] and LLaMA 3 [3] have rapidly penetrated the personal and professional domains, transforming workflows in academic, industrial and everyday applications.

Despite their success, 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. Although 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) methods [4], such as Low-Rank Adaptation (LoRA) [5], have become prevalent. LoRA and its derivatives (e.g., QLoRA [6], LoRA + [7]) reduce computational overhead by using reparametrized weight matrices, where large weight matrices are approximated with lower-rank decomposition. This approach enables fine-tuning even on limited-hardware setups.

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[8], along with classical parameters such as learning rate and weight decay.

Hyperparameter Optimization (HPO) is a well-studied problem [9,10,11] in Automated Deep Neural Networks (autoDNN) design [12]. 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 process, often requiring hours to days on high-performance hardware. These constraints necessitate the adoption of highly efficient optimization techniques.

Bayesian Optimization (BO) [13] 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 is by nature sequential, limiting the use of HPC cluster, or exascale computer. In contrast, Partition Based Optimization (PBO) methods [14,15],

which divide the search space into subregions, enable parallel evaluations but typically require more function calls to converge. The complementarity of these methods made way for hybrid algorithms, using surrogate to enhance PBO algorithms.

In this work, we present the HPO problem for LLM fine-tuning, to address it with Bayesian Optimization and Partition Based Optimization methods. By comparing these methods, and trying the hybridization, we aim to open the way for parallel Bayesian algorithm for optimization of very expensive objective function with exascale computing.

2 Problem definition

Hyperparameter Optimization (HPO) of LLM Fine-Tuning is a very recent problem in literature, with a first review on article [16], but HPO for Deep Neural Networks (DNN) is a known field with the autoML community. For further reading, Talbi write a taxonomy of autoDNN in article [12].

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 constrains the sets of methods to be used directly. Following article [17] notation, this problem is also a Mixed-Variables Optimization Problem (MVOP), w.r.t. the set of values possible for each hyperparameter. The last key aspect of this optimization problem is the cost of evaluating a solution, it can take dozens of minutes to hours, restraining the number of possible evaluations.

Given a model $\mathcal{M}$, 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 linking 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 $\mathcal{F}(\cdot, \cdot, \cdot, \cdot)$.

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

2.1 Search Space

The search space is defined with the choice of hyperparameters, theirs bounds, theirs types and even theirs scales. 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.

$$\begin{aligned} W = W_0 + \Delta W = W_0 + \frac{\alpha}{r}(B.A) \\ s.t. \quad W, W_0, \Delta W \in \mathbb{R}^{n*p}, A \in \mathbb{R}^{r*p} \text{ and } B \in \mathbb{R}^{n*r} \end{aligned} \tag{2}$$

The search space is composed of 5 hyperparameters, from classical training hyperparameters to LoRA specific ones. Equation 2 summarize LoRA application, w.r.t. it's hyperparameter, with W_0 the weights of the pre-trained model and $W = \Delta W$ the additional gradients to add to obtain W the weights of the fine-tuned model. A detailed presentation of hyperparameters is just below, but one can look at table 1 for a summary.

- LoRA rank : the common rank of matrices A and B, scaling the reduction in terms of number of parameters. It's an integer, and it's value range from 1 to 64 to deal with hardware constraints.
- LoRA scale (α) : α is used to scale the values of $B * A$ when added to W_0. On this work, due to LitGPT framework, it's an integer, from 1 to 64.
- Learning rate : the learning rate is a classical hyperparameter used in HPO, weighting the gradient of each weight when doing back-propagation. It is often tuned in a logarithmic scale, to manage effectively the exploration. Hyperparameter value is between 10^{-10} and 10^{-1}.
- Dropout probability : based on article [18], 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 [19], 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}.

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

Table 1: Summary of Hyperparameter Search Space

For the 2 integers variables (LoRA rank and LoRA scale), to adapt to continuous optimization algorithms, relax and round methods will be applied. It means 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

Classically 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, and 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 [20] 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[21] or MMLU[22,23]. In this paper, 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 field, 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 [12] 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 and the evaluation of the model. 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 and EA, a lot of articles [24,25,26] 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 [27,28] aims to optimize the prompt, and work with the entire LLM as a generating black box, without thinking about the architecture or the weights. This approach has limited effects on LLM, since the performance are still bound by model training, and they replace the end users for chatbot deployment.
– HPO applied to generation hyperparameter : some hyperparameters like the *temperature* are only used when LLM are generating tokens. Some are working on these hyperparameters, to avoid training costs of working directly with the fine-tuning hyperparameters.

3 Design and Implementation

In this work, two approaches are considered to perform Hyperparameter Optimization (HPO) : Bayesian Optimization based on Gaussian Process (BO-GP) and Partition Based Optimization (PBO) algorithms like Simultaneous Optimistic Optimization (SOO)[15]. To extract the best of these two approaches, a hybrid approach will also be presented.

Figure 1 describe the global workflow of Hyperparameter Optimization (HPO), to link with following sections aiming to describe sections of the workflow.

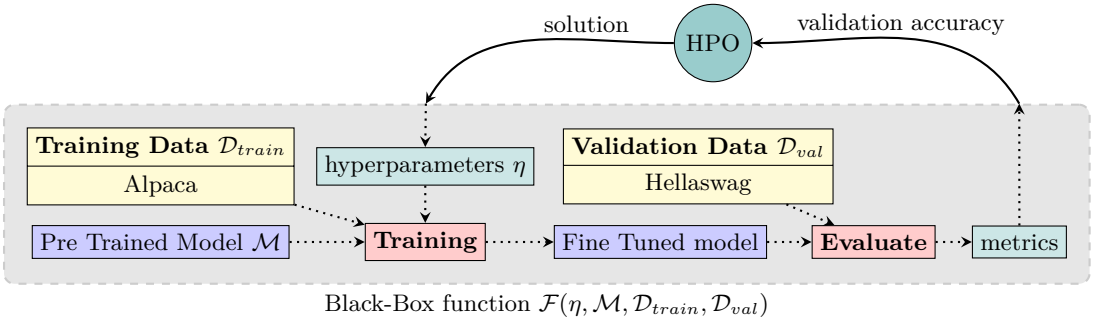


Fig. 1: HPO workflow

The workflow is budget-based, meaning all algorithms will be compared with the same number of evaluations, i.e. the budget. This approach is effective because it ensures a fair comparison by focusing on the evaluation cost, which is the primary bottleneck in optimization algorithms, rather than computational overhead, which is negligible in this context. 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 for specific black-box function, and must adequate with the characteristic of this objective function. As said in section 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 3.2-1B [3], 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 performed using AdamW [29] 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 matrices, enabling the training of whole Multi-Head Attention (MHA) 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 [6,30,31] for fine-tuning, guiding our choice for a standard configuration.

For evaluation, Hellaswag [21] dataset was used as $\mathcal{D}_{val}$, with accuracy as the metrics to optimize with HPO. It's a 40k lines datasets released in 2019 as a challenge datasets, with a random performance of 25 % . All models are also evaluated on MMLU [22] as a testing dataset, to observe HPO over-fitting.

In terms of framework, all this part is done using LitGPT [32], a framework develop by Lightning AI team, based on Pytorch [33] 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 for training the model, Huggingface for loading model and datasets, and lm_eval library 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" [13]. A surrogate model is used to build a posterior considering prior knowledge formed on known points. On this posterior, an acquisition function is computed to act as the surrogate function for the function to optimize. Many surrogate models can be used like regression tree [34] or Gaussian Process (GP) [35]. For further details about BO, one can read review [13].

On this work, a focus is done on GP for the BO surrogate. GP use the kernel trick to build a Bayesian non-parametric regression model. It uses a mean vector m_i and a covariance matrix $K_{i,j}$ to define the prior function as equation 3.

$$\mathrm{f}|X \sim \mathcal{N}(m, K) \tag{3}$$

Algorithm 1 BO-GP

Require: Ω,f,K_D,$\mathcal{O}$,f_{acq},n_{init},n_{opt}
1: **for** $i = 1$ **to** n_{init} **do** ▷ Initiate with Latin Hypercube Sampling
2: $\lambda' \leftarrow LHS(\Omega, \mathcal{D}, n_{init})$ ▷ 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: **for** $i = 1$ **to** n_{opt} **do** ▷ Optimization loop
6: $K_D, \mu_D \leftarrow \text{Fit}(\text{GP}(K_D, \mu_D), \mathcal{D})$
7: $\lambda' \leftarrow \text{Optimize}(f_{\text{acq}}(K_D), \mathcal{O})$ ▷ Generate new point
8: $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\lambda', f(\lambda'))\}$ ▷ Evaluate new point
9: **end for**
10: **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 [36], to sample efficiently the search space. This sampling provides information for the Gaussian Process to estimate the function. Like article [37], 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 loop containing the update of the Gaussian Process, the optimization of the acquisition function to obtain a new point to evaluate and the evaluation. 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}, with the budget $n_{max} = n_{init} + n_{opt}$

For this algorithm, the first requirement 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 [38]. 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 [39] 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 [40] for Gaussian Processes, BoTorch enables advanced surrogate modeling and optimization. Botorch is used on this work for all tasks using GP, including 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 [14,41,15]. These approaches are mostly deterministic, and enhance intrinsic parallelization ability. For these 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 [42], the Simultaneous Optimistic Optimization (SOO) [15] algorithm seems a good way to start.

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

Algorithm 2 SOO

Require: Ω,f,K,n_{max}
1: $x_{0,0} \leftarrow center(\Omega)$,$f_{0,0} \leftarrow f(x_{0,0})$,$\mathcal{T}_1 = \{x_{0,0}, f_{0,0}, \Omega\}$ ▷ Initiate the tree with the center of Ω
2: $n \leftarrow 1$
3: **while** $n < n_{max}$ **do**
4: $\nu_{max} \leftarrow -\infty$
5: **for** $h = 0$ **to** $depth(\mathcal{T}_n)$ **do**
6: $j \leftarrow \arg\max_{j \in \{j|(h,j) \in L_n\}} f(x_{h,j})$ ▷ Select best open leaf for depth h
7: **if** $f_{h,j} > \nu_{max}$ **then**
8: $\Omega_{h+1,kj+1}, \ldots, \Omega_{h+1,kj+K} \leftarrow section(\Omega_{h,j}, K)$ ▷ perform K-section of $\Omega_{h,j}$
9: **for** $i = 1$ **to**K **do**
10: $n \leftarrow n + 1$
11: $x_{h+1,kj+i} \leftarrow center(\Omega_n)$
12: $f_{h+1,kj+i} \leftarrow f(x_{h+1,kj+i})$ ▷ evaluate the point, the scoring
13: $\mathcal{T}_n \leftarrow \{(x_{h+1,kj+i}, f_{h+1,kj+i}, \Omega_{n+1})\}$ ▷ Add leaf$_{h+1,j+i}$ to tree $\mathcal{T}_n$
14: **end for**
15: $\nu_{max} \leftarrow f_{h,j}$
16: **end if**
17: **end for**
18: **end while**
19: **return** best of $x_{h,j}, f(x_{h,j})$

such that $f(x^*) - f(x) \leq l(x, x^*)$ where x^* is the maximizer of x. The algorithm partitions the space Ω by building a tree with smaller and smaller subspace $\Omega_{h,j}$. The node (h, j), 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, $K = 3$ here. 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 manages 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 information to define a cost-reduced function to optimize. This approach ensure exploitation but have several limitations, including the intrinsic sequential nature. 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. Like SOO, BaMSOO performs a K-inary partitioning of the space, using the center of the partition to evaluate.

$$\begin{aligned} \mathcal{UCB}(x|\mathcal{D}_t) &= \mu(x|\mathcal{D}_t) + B_N * \sigma(x|\mathcal{D}_t) \\ \text{with } B_N &= \sqrt{2\log(\pi^2 N^2/6\eta)}, \eta \in (0,1) \end{aligned} \tag{4}$$

The difference lies primarily in the scoring $g(.)$ of the leaf, with algorithm 3 replacing the scoring of SOO (line 12 of algorithm 2). In the face of an expensive objective function, BaMSOO leverages a GP surrogate to estimate the potential of a point, using the Upper Confidence Bound ($\mathcal{UCB}$) as a measure of expected performance.

Given a partition with center x and historic evaluations $\mathcal{D}_t$, the $\mathcal{UCB}$ of x, defined in Equation 4, is compared against the best evaluation so far, f^+. In this equation, η is a hyperparameter to define manually, and N the number of evaluations. 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 Lower Confidence Bound (LCB) of x, reflecting the lower bound of potential improvement.

Algorithm 3 BaMSOO Scoring

if $\mathcal{UCB}(x_{h+1,kj+i}, \mathcal{D}_N) \geq f^+$ **then** ▷ if x may be better than previous score
  $g_{h+1,kj+i} \leftarrow f(x_{h+1,kj+i})$ ▷ Evaluate x
  $N \leftarrow N + 1$ ▷ index of the number of *real* evaluation
else
  $g_{h+1,j+i} \leftarrow \mathcal{LCB}(x_{h+1,kj+i}, \mathcal{D}_N)$ ▷ Penalize with LCB
end if
if $g_{h+1,j+i} > f^+$ **then**
  $f^+ \leftarrow g_{h+1,j+i}$ ▷ f^+ is the highest score of the tree
end if

To sum up, this algorithm prevents unpromising evaluations in order to allocate more budget for exploring more promising areas than SOO. This hybrid approach harness a part of BO-GP exploitation of knowledge without losing the intrinsic parallel abilities.

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 implementation outlined in section 3.2, the framework ensures consistency in surrogate modeling and acquisition function computation.

4 Computation Experiments

Experiments are done using Grid5000 [43]," 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 (weight matrix to apply LoRA), all arguments of LitGPT CLI is used with default value. The only exception is the number of epochs, fixed to 1 to reduce computation costs. 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 around 31 minutes. For the evaluation on both datasets, it's taking around 12 minutes. Based on previous articles, and evaluations durations, the total evaluation budget for experiments is 50 evaluations by each algorithm, including a sampling budget of 10 for Bayesian Optimization.

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

4.1 Sampling experiment

For the sampling experiment, Latin Hypercube Sampling (LHS) was used with the same budget as other, i.e. 50 evaluations. Whole experiments took 36 hours. The aims of these experiments is dual: first, make a lower bound reference for others experiments, and second, explore the search space and the score behavior. To use as a lower bound, LHS achieve scores of 37.6% for MMLU, and 47.9% for Hellaswag. These score are inside table 2.

The correlation between MMLU and Hellswag tend to confirm the relevance of their choice as a pair for validation and testing dataset. It's sufficiently high to guess that most of the *good* solution for one will be good for the other one, but it's still less than one, making over-fitting visible if present.

For variables, it looks like LoRA alpha, the scaling parameter of LoRA, is the most influent on the score, be it Hellaswag or MMLU. Next to it, LoRA Rank and weight decay are the most influent, with a difference in ranking on the metrics. With this first exploration, it looks like Dropout and Learning rate are not very effectful for this problem, be it the choice of their range or the implementation.

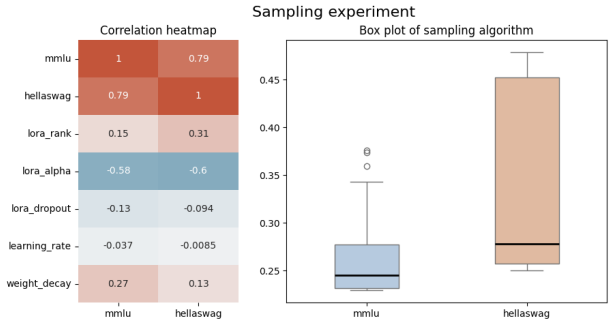
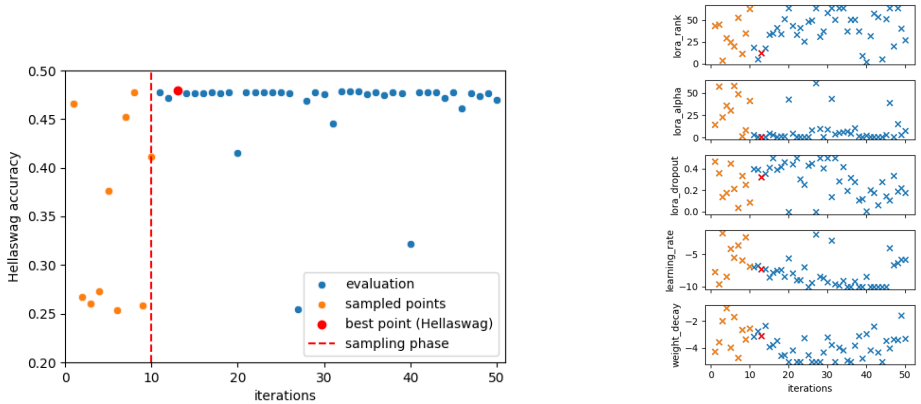


Fig. 2: Results of Latin Hypercube Sampling experiment

On the distribution of the score values, it's interesting to note that Hellaswag has a broader range of score, making it useful for efficiently discriminating solutions. With a broader thinking, the sampling experiment confirm the relevance to apply HPO algorithms to the described problem. If scores were independent of variables, or all scores were the same, HPO would be useless.

4.2 BO-GP experiment

Figure 3 depicts the performance of Bayesian Optimization based on Gaussian Process (BO-GP) over 50 iterations, measured in terms of the Hellaswag accuracy. 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 over time

Fig. 3: Experiment using BO algorithm

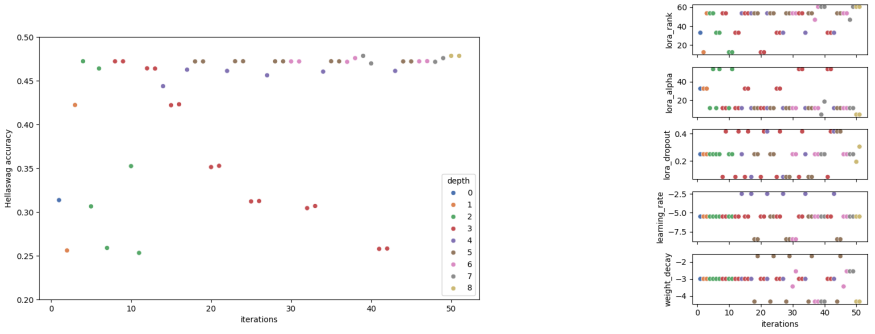
During the sampling phase, as shown by figure 3a, evaluated solutions have the same diversity of solutions than the full sampling experiment, allowing to efficiently extract knowledge from the search space. The algorithm achieves a stable convergence to mostly high-performing solution.

If we briefly look at figure 3b, it's interesting to look at the diversity of configuration to obtain high-performance, and the trend of the algorithms for each variable. For instance, for LoRA alpha, or learning rate, we can observe a clear trend toward the lower bound of the optimization range.

To summarize, this experiment demonstrate the effectiveness of Bayesian Optimization in efficiently exploiting the knowledge of the search space obtained in the sampling phase. The optimization process still explores area with high uncertainty, giving few low-performing solution.

4.3 SOO experiment

On this experiment, we observe in Figure 4 the behavior of SOO as it optimizes the given objective function. The figure illustrates how the algorithm navigates the search space, iteratively improving the solution by adjusting the hyperparameters. By examining the trends in performance metrics and variables evolution, we aim to analyze the convergence behavior, stability, and overall effectiveness of SOO in this setting.



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

Fig. 4: Experiment using SOO algorithm

Figure 4a depicts the accuracy of the hellaswag dataset as a function of iterations, with marker colors representing different depth configurations. The observed trend suggests that most configurations achieve accuracies around 0.46 throughout the optimization process. Several configurations converge toward the higher end of this range, indicating potential stability and effectiveness in learning as iterations progress.

It is evident that higher-depth configurations tend to cluster around higher accuracy values, while lower depths display more scattered behavior across the accuracy spectrum. In specific, depth 2, with a section on lora alpha seems to greatly affect the performance on the accuracy. On figure 4b, it's clear on the limitation of constrained-budget SOO without local exploitation at the end of the algorithms : Only a few number of values by variables are explored : around 5 by variables, making it equivalent to exploration of a search space of $6^5 = 7,776$ discrete solutions.

To summarize, SOO achieve a maximum performance closed to the precedent algorithm, but explore a restricted number of configuration, and lose time by evaluating unpromising solutions.

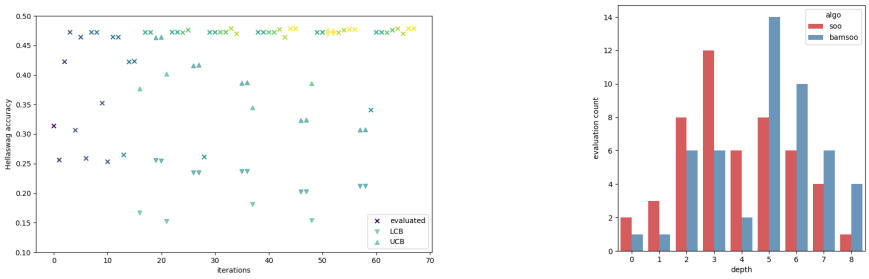
4.4 BaMSOO experiment

The last experiment of this work is using BaMSOO, with $\eta = 1$ in equation 4, to reduce the confidence interval and promote approximation, to avoid the evaluation of unpromising points. To look after it, figure 5a discriminate evaluated and approximated points, and might be compared with 4a to look after approximated points.

When compared to figure 4a, the pair of evaluations for depth 2, with low-performing solutions seems to have mostly been approximated. On the whole experiment, 16 points were approximated, to efficiently search further than SOO. Figure 5b compare the budget allowed on each depth, and highlight the focus of BaMSOO for lower depth exploration. To summarize, BaMSOO achieve to speed up SOO, but is still slow in comparison with BO at first glance. Moreover, BaMSOO succeed to prevent most of low-performing evaluation of SOO.

4.5 Comparison and Analysis

To conclude this section, it's crucial to directly compare algorithms, especially their performance, with validation and testing dataset. To look at absolute performance, and not only relative between algorithms, lower and upper bounds will be used.



(a) Score over time (b) Depth exploration

Fig. 5: Experiment using BaMSOO algorithm

Datasets	Lower (LHS)	Upper (model card)	BO-GP	SOO	BaMSOO
Hellaswag (validation)	47.90	41.5	47.91	47.84	47.84
MMLU (testing)	37.61	49.3	38.11	37.42	37.50

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 are the results of the experiment using solely LHS to pick solutions, with the same number of evaluation than algorithms. LHS being intrinsically parallel, if evaluated algorithms don't achieve better performance than a solely exploring one, their benefit isn't relevant.

For the higher bound, I will look at the higher value in the model card[1] of the model, achieved using advanced fine-tuning methods like Supervised Fine-Tuning (SFT), Rejection Sampling (RS), and Direct Preference Optimization (DPO). Values are inside table 2.

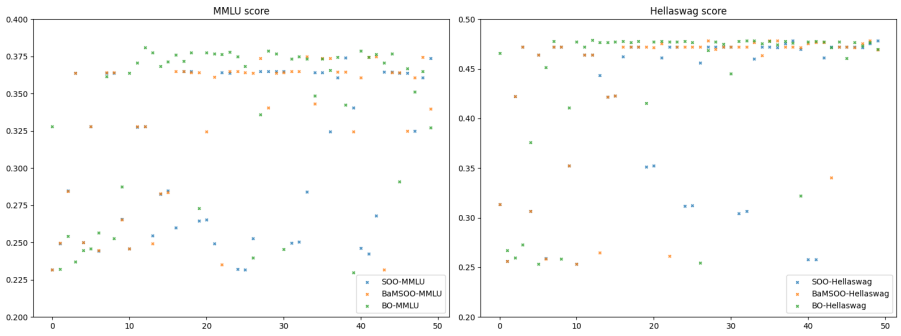


Fig. 6: Comparison between 3 algorithms on 2 metrics

At first, what's interesting is to look at testing and upper bounds results. Since the upper bound is mostly fine-tuned with aimed for MMLU results, the Hellaswag upper bounds isn't really relevant. Considering this, the analysis of this part will mostly be done with MMLU. The best results with MMLU is Bayesian Optimization (BO), being 23% from the upper bound.

With figure 6, it's clearer than BO algorithm is a step more performing than others algorithms for this problem, especially with the low number of low-performing solutions evaluated. Apart from this part, the whole results may suggest that the objective function, or the search space could be different, to have an higher range of results for comparison.

[1] `https://huggingface.co/meta-llama/Llama-3.2-1B`

5 Conclusions and Perspectives

For scenarios with a limited number of evaluations, BO demonstrated exceptional efficiency, quickly converging to high-performing solutions. In contrast, SOO, constrained by one-dimensional sections and less promising search dynamics, exhibited slower convergence. The hybrid approach, BaMSOO, successfully enhanced the convergence rate of SOO while preserving its inherent parallelism capabilities.

This work pioneers the application of hybrid Bayesian and partition-based optimization for expensive functions, laying the groundwork for scaling the efficiency of Bayesian Optimization to exascale computing environments.

Future research should explore higher-performance architectures, larger models, and expanded datasets to effectively distribute the computational load across supercomputers. Additionally, extending the search space to include hyperparameters such as Adam momentum would provide new opportunities to further refine optimization strategies.

Acknowledgments

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. Vaswani, A., Shazeer, N., Parmar, N. et al.: Attention is All you Need. In: Advances in Neural Information Processing Systems. Volume 30., Curran Associates, Inc. (2017)
2. OpenAI, Achiam, J., Adler, S. et al.: GPT-4 Technical Report (2024) arXiv:2303.08774 [cs].
3. Grattafiori, A., Dubey, A., Jauhri, A. et al.: The Llama 3 Herd of Models (2024) arXiv:2407.21783 [cs].
4. Han, Z., Gao, C., Liu, J. et al.: Parameter-Efficient Fine-Tuning for Large Models: A Comprehensive Survey (2024) arXiv:2403.14608 [cs].
5. Hu, E.J., Shen, Y., Wallis, P. et al.: LoRA: Low-Rank Adaptation of Large Language Models (2021) arXiv:2106.09685.
6. Dettmers, T., Pagnoni, A., Holtzman, A. et al.: QLoRA: Efficient Finetuning of Quantized LLMs. Advances in Neural Information Processing Systems **36** (2023) 10088–10115
7. Hayou, S., Ghosh, N., : LoRA+: Efficient Low Rank Adaptation of Large Models (2024) arXiv:2402.12354 [cs].
8. 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].
9. Bischl, B., Binder, M., Lang, M. et al.: Hyperparameter Optimization: Foundations, Algorithms, Best Practices and Open Challenges (2021) arXiv:2107.05847 [stat].
10. Bergstra, J., Bardenet, R., Bengio, Y. et al.: Algorithms for Hyper-Parameter Optimization. Volume 24., Neural Information Processing Systems Foundation (2011)
11. Feurer, M., : Hyperparameter Optimization. In Hutter, F., Kotthoff, L., , eds.: Automated Machine Learning: Methods, Systems, Challenges. Springer International Publishing, Cham (2019) 3–33
12. Talbi, E.G.: Automated Design of Deep Neural Networks: A Survey and Unified Taxonomy. ACM Comput. Surv. **54** (2021) 34:1–34:37
13. 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.
14. 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
15. 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)
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. Srivastava, N., Hinton, G., Krizhevsky, A. et al.: Dropout: A Simple Way to Prevent Neural Networks from Overfitting. (2014)
19. Krogh, A., : A Simple Weight Decay Can Improve Generalization. In: Advances in Neural Information Processing Systems. Volume 4., Morgan-Kaufmann (1991)
20. Wei, J., Bosma, M., Zhao, V.Y. et al.: Finetuned Language Models Are Zero-Shot Learners (2022) arXiv:2109.01652 [cs].
21. Zellers, R., Holtzman, A., Bisk, Y. et al.: HellaSwag: Can a Machine Really Finish Your Sentence? (2019) arXiv:1905.07830 [cs].
22. Hendrycks, D., Burns, C., Basart, S. et al.: Measuring massive multitask language understanding. Proceedings of the International Conference on Learning Representations (ICLR) (2021)
23. Hendrycks, D., Burns, C., Basart, S. et al.: Aligning ai with shared human values. Proceedings of the International Conference on Learning Representations (ICLR) (2021)
24. 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
25. 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
26. Brahmachary, S., Joshi, S.M., Panda, A. et al.: Large Language Model-Based Evolutionary Optimizer: Reasoning with elitism (2024) arXiv:2403.02054 [cs].
27. Diao, S., Huang, Z., Xu, R. et al.: Black-box Prompt Learning for Pre-trained Language Models (2023) arXiv:2201.08531 [cs].
28. Xu, H., Chen, Y., Du, Y. et al.: GPS: Genetic Prompt Search for Efficient Few-shot Learning (2022) arXiv:2210.17041 [cs].
29. Loshchilov, I., : Decoupled Weight Decay Regularization (2019) arXiv:1711.05101 [cs].
30. Chung, H.W., Hou, L., Longpre, S. et al.: Scaling Instruction-Finetuned Language Models. Journal of Machine Learning Research **25** (2024) 1–53
31. Zhou, C., Liu, P., Xu, P. et al.: LIMA: Less Is More for Alignment. Advances in Neural Information Processing Systems **36** (2023) 55006–55021
32. The Lightning AI team: LitGPT (2023) original-date: 2023-05-04T17:46:11Z.
33. 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.
34. 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
35. 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.
36. 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.
37. 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.
38. Ament, S., Daulton, S., Eriksson, D. et al.: Unexpected Improvements to Expected Improvement for Bayesian Optimization (2024) arXiv:2310.20708 [cs].
39. Balandat, M., Karrer, B., Jiang, D.R. et al.: BoTorch: A Framework for Efficient Monte-Carlo Bayesian Optimization (2020) arXiv:1910.06403 [cs].
40. Gardner, J.R., Pleiss, G., Bindel, D. et al.: GPyTorch: Blackbox Matrix-Matrix Gaussian Process Inference with GPU Acceleration (2021) arXiv:1809.11165 [cs].
41. Jones, D.R., Perttunen, C.D., : Lipschitzian optimization without the Lipschitz constant. Journal of Optimization Theory and Applications **79** (1993) 157–181
42. Firmin, T., : A fractal-based decomposition framework for continuous optimization. (2022)
43. Balouek, D., Carpen-Amarie, A., Charrier, G. et al.: Adding Virtualization Capabilities to Grid'5000. report, INRIA (2012) Pages: 18.