Is One Epoch All You Need For Multi-Fidelity Hyperparameter Optimization?

Romain Egelé[1], Isabelle Guyon[2], Yixuan Sun[3] and Prasanna Balaprakash[4]

1- LISN, Université Paris-Saclay, France & MCS, Argonne National Laboratory, USA
2- LISN/Inria/CNRS, Université Paris-Saclay, France & Google, USA & ChaLearn, USA
3- MCS, Argonne National Laboratory, USA
4- CCSD, Oak Ridge National Laboratory, USA

Abstract. Hyperparameter optimization (HPO) is important for adjusting machine learning models, but it can be computationally expensive. Multi-fidelity HPO (MF-HPO) reduces cost by using intermediate accuracy and discarding low-performing models. Many such methods have been proposed, but it is unclear which is most effective. Using different benchmarks, we compare a simple baseline training for 1-Epoch against popular methods. Results showed unexpected good performance of the baseline reaching similar accuracy with approximately ten times fewer training epochs. The learning curves from these benchmarks are analysed and suggest increasing the diversity of MF-HPO benchmarks to better include "short-term horizon bias" cases.

1 Introduction

Hyperparameter optimization (HPO) is a key component of AutoML systems. It aims to find the best configuration of a machine learning pipeline, which consists of data and model components. Hyperparameters are parameters that cannot be learned from the data, but affect the learning process and the performance of the pipeline. HPO can be seen as a black-box problem[1, 2], where a function maps a hyperparameter configuration to a performance score. However, evaluating this function can be costly and time-consuming. Therefore, MF-HPO methods have been proposed, which use intermediate performance estimates to reduce the computational cost and speed up the optimization process.

To set the stage, first, we present the generic components of muti-fidelity HPO in Figure 1 which is similar to previous work on neural architecture search [3]. The input to HPO is the hyperparameter search space. The output is a trained machine-learning pipeline. HPO is a bi-level optimization problem: the upper level seeks to optimize the hyperparameters of the ML pipeline and the lower level seeks to optimize the parameters of the learning model for a given hyperparameter configuration from the upper level. In Figure 1, the outer loop (upper-level problem) explores hyperparameter configurations while the inner loop (lower level problem) evaluates these configurations by exploring their performance at different training steps. When finishing the search, a final model selection step is performed before returning the trained algorithm.

In regular HPO, the inner loop is often fixed with a given stopping criteria and therefore seen as a black box. A natural refinement is now to consider the multi-fidelity setting in which estimations of the black-box are obtained at various steps of the inner optimization such as the training samples or the training epochs, which boils down to observing a "learning curve".

Fig. 1: Hyperparameter optimization and its components including input/output, outer optimization loop exploring new HP configurations, inner optimization loop incrementally allocating new training "steps" and model selection.

In this study, we focus on evaluating the effectiveness of early discarding policies for the inner loop. For the outer loop we consider random search in our experiments. More precisely for early discarding, we consider the most popular and widespread approaches with zero initialization cost. Two baselines, at each extreme, are considered. One evaluates each hyperparameter configuration candidate at maximum fidelity while the other evaluates at minimum fidelity (1-Epoch). We show the unreasonable **effectiveness of the 1-Epoch approach** and characterize the reason for its success. We **argue for including such a simple baseline** for the evaluation of early discarding policies **in the MF-HPO studies**.

2 Related Work

Our study focuses on early discarding with vertical model selection as described in [4]. Suggested candidates are streamed one by one and one should decide how to grow their learning curves such as in Asynchronous Successive Halving [5] (SHA). Some noticeable methods which can be adapted to this setting are: Hyperband [6] (HB) which can explore levels of fidelity to differentiate noisy learning curves; Freeze-Thaw Bayesian optimization [7] which can pause/continue the training of a model and optimize the hyperparameter configuration; Learning curve extrapolation [8] (LCE) which can observe early performances and extrapolate future performances to decide if training should continue; FABOLAS [9] which learns correlations of the candidates ranking between levels of fidelity; Bayesian Optimization Hyperband [10] which embeds Bayesian optimization in Hyperband to sample candidates more efficiently; Learning Curve with Support Vector Regression [11] which predicts final performance from the configuration and early observations; Learning Curve with Bayesian neural network [12] which is similar to the previous one but switching the model with a Bayesian neural network;

and trace aware knowledge-gradient [13] which leverages an observed curve to update the posterior distribution of a Gaussian process more efficiently.

A first set of limitations in previous works is due to assumptions on the learning-curve. For methods based on SHA, it assumes that discarded learning curves do not cross in the future because only the Top-K are allowed to continue at a given step. Therefore, models with a slow start can be discarded (often related to configurations of the search space with slower convergence). This phenomenon is also refered to as "short-horizon bias" [14]. For methods based on LCE, it either assumes a "biased" parametric model or it is more generic but require a more expensive initialization [15]. A second limitation is that the returned configuration is is often the lowest validation error observed during the search. However, this is very limiting because it does not consider more sophisticated model selection scheme (cross-validation, Hyperband) which is why we added the selection block in Figure 1.

Benchmarks play a crucial role in the design and development of HPO methods. We surveyed benchmarks for learning curves which are all quite recent at the time of writing and keep evolving: HPOBench [16, 17], LCBench [18], LCDB [19], NAS-Bench-x11 [20], JAHS-Bench-201 [21] and YAHPO-Gym [22].

3 Method

In this section, we present the baseline using the minimum fidelity (i.e., optimal budget utilization) and other multi-fidelity agents.

The 1-Epoch baseline for the inner loop optimization comprises two phases: The first **search phase** focuses on exploiting early feedback from training with the minimum number of training iterations (1 Epoch; minimum fidelity) (blue loop in Figure 1). The second **model selection phase** focuses on exploring the performance of the most promising candidates from the first phase by evaluating them at the maximum fidelity.

We selected other multi-fidelity agents (inner-optimization loop) based on their popularity (citations, implementation in packages) and also because they require zero-initialization cost [15]—they do not require pre-training on learning curve data. The first method considered is Successive Halving (SHA) [23] which initially assumed a fixed set of learners but is now generalized to a non-limited set (therefore can handle a stream) in an asynchronous variant [5]. SHA decides to continue the training loop at exponentially increasing steps called rungs (reflecting the theory being learning curves [24]). At each rung, the algorithm will check if the current observed score is among the Top-K scores and continue if the condition is met, discard otherwise. Doing so can be dangerous as learning curve scores are noisy. The 1-Epoch strategy would suffer even more than SHA in this situation. To make SHA more robust in this setting the Hyperband (HB) approach [6] proposes to enforce the exploration of more levels of fidelity on top of SHA to reduce the effect of the impact of the noise in the decision. As a result of this exploration of fidelities, HB is consuming more training steps than SHA by construction but is also less likely to be stuck with underperforming

hyperparameter configurations. Finally, the last approach we consider consists in using a surrogate model to predict the future of the learning curve and is therefore called learning curve extrapolation (LCE) [8]. In theory, this approach is less likely to get stuck with under-performing configurations like SHA or HB could in the case where the best models at the last training step would not be among the best at the first training step (slower convergence of the learning curve). After observing some instabilities on the implementation from [8] we decide to make the approach more robust and proposed a faster and more stable implementation of the model which we call RoBER (detailed in our extending set of materials).

4 Experimental Results

First we introduce our experimental setting. Then, we evaluate our different multi-fidelity strategies and characterise benchmarks properties explaining the unreasonable effectiveness of this baseline.

In our analysis we looked at four benchmarks: HPOBench, LCBench, YAHPO-Gym and JAHS-Bench. But, as we observed similar results, for brevity we only expose the Naval Propulsion problem from HPOBench. We provide complementary materials [1] with our full set of results.

Fig. 2: Comparing the performance (mean and standard error) of various early discarding strategies (all combined with with random search) on the Naval Propulsion problem from the HPOBench benchmark.

We set the number of iterations for the outer-loop optimization to 200 (red loop in Fig. 1). For visualization, we show the evolution of the test error with respect to the number of training epochs consumed by the algorithm [22]. It assumes that each step is equally costly which is not always the case but our take is that reducing the number of consumed training epochs is more important overall to advance the multi-fidelity field. As a result, the base policy, which evaluates each candidate at maximum fidelity (100-epochs) consumes the maximum number of 20,000 training epochs. Similarly, its counterpart at minimum fidelity (1-Epoch, performed with a Top-3 model selection ($K = 3$) at maximum fidelity after the search), consumes a fixed number of $200 \times 1 + 3 \times 100 = 500$ training epochs. Each HPO run is performed with 10 different random states and we present the average results along with their standard error.

The evolution of the test score with respect to the number of consumed training epochs is shown in Figure 2. As it can be observed, the 1-Epoch policy consumes 40 times fewer training epochs than its counterpart (100-Epochs) as explained in our experimental setup. We can also see that the test objective found is very

[1] `https://github.com/deephyper/scalable-bo/blob/main/esann-23/One_Epoch_Is_Often_All_You_Need_Extended.pdf`

(a) Learning curves colored by final rank.

(b) Evolution of the ranking with epochs.

Fig. 3: Visualizing the ranking for good (blue) and bad (red) models for 1,000 randomly sampled learning curves from the Naval Propulsion benchmark. In (a) it shows that models can be selected from the first epoch. In (b) it shows that the ranking stabilizes faster among the best models.

close to the test objective of the 100-Epochs strategy. Furthermore, we observe that more complex agents (SHA, HB, and LCE) are not significantly different from each other, with the exception that Hyperband, which always consumes slightly more resources than SHA by design. SHA is the agent consuming the least training epochs with about 10 times fewer epochs than 100-Epochs but 4 times more than 1-Epoch.

Now that we have seen the performance of training for only 1 epoch during the search, we will explain the reasons behind it. In Figure 3a, we display 1,000 randomly sampled learning curves, and color the curves by their ranking at maximum fidelity. A low rank colored in blue corresponds to a good model, similarly, a high rank colored in red corresponds to a bad model. From this visualization, the groups of bad and good models can already be identified at the first epoch of training. This explains why selection with 1 epoch is effective. Also, as can be seen for the blue curves, some noise exists between the models, which explains why the top-K tournament selection is necessary and effective to distinguish among different models (following the idea of Hyperband, which increases the number of epochs to differentiate noisy scores). The evolution of the noise in the ranking for the same benchmark can be visualized in Figure 3b. In this figure, we display the ranking for the same models at each training epoch. It is clear that the first half (before 50 epochs) has more noise (crossing learning curves) than the second half. It can also be seen that the ranking of good models stabilizes faster, already after 5 epochs, compared to the ranking of bad models which remains relatively noisy until the end.

5 Conclusion

This study investigated how to optimize model's hyperparameters with minimal training steps using MF-HPO agents. We compared different multi-fidelity policies and found that the 1-Epoch strategy, which evaluates each candidate configuration at the lowest fidelity, performed well on common benchmarks. This is attributed to the fact that good and bad models can be distinguished early on in training.

We found that this phenomenon is not unique to the presented benchmark, but occurs in other common benchmarks from the litterature. We think that easy and difficult benchmarks should be better caracterized by including such a simple baseline. Despite these favorable results, a limitation of our work is to consider "epoch" as the unit of fidelity. For very large dataset this could be too costly (e.g., large text corporas used to trained language models). Similarly, for very small datasets it could not be too little data.

References

[1] J. Bergstra et al. Algorithms for hyper-parameter optimization.

[2] T. Yu et al. Hyper-parameter optimization: A review of algorithms and applications.

[3] T. Elsken et al. Neural architecture search: A survey.

[4] F. Mohr et al. Learning curves for decision making in supervised machine learning – a survey.

[5] L. Li et al. A system for massively parallel hyperparameter tuning.

[6] L. Li et al. Hyperband: A novel bandit-based approach to hyperparameter optimization.

[7] K. Swersky et al. Freeze-thaw bayesian optimization.

[8] T. Domhan et al. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves.

[9] A. Klein et al. Fast bayesian optimization of machine learning hyperparameters on large datasets.

[10] S. Falkner et al. BOHB: Robust and efficient hyperparameter optimization at scale.

[11] B. Baker et al. Accelerating neural architecture search using performance prediction.

[12] A. Klein et al. Learning cruve prediction with bayesian neural networks.

[13] J. Wu et al. Practical multi-fidelity bayesian optimization for hyperparameter tuning.

[14] Y. Wu et al. Understanding short-horizon bias in stochastic meta-optimization.

[15] C. White et al. How powerful are performance predictors in neural architecture search?

[16] K. Eggensperger et al. HPOBench: A collection of reproducible multi-fidelity benchmark problems for HPO.

[17] A. Klein et al. Tabular benchmarks for joint architecture and hyperparameter optimization.

[18] L. Zimmer et al. Auto-PyTorch tabular: Multi-fidelity MetaLearning for efficient and robust AutoDL.

[19] F. Mohr et al. Lcdb 1.0: An extensive learning curves database for classification tasks.

[20] Y. Shen et al. Nas-bench-x11 and the power of learning curves.

[21] A. Bansal et al. JAHS-bench-201: A foundation for research on joint architecture and hyperparameter search.

[22] F. Pfisterer et al. Yahpo gym-an efficient multi-objective multi-fidelity benchmark for hyperparameter optimization.

[23] K. Jamieson et al. Non-stochastic best arm identification and hyperparameter optimization.

[24] M. Hutter. Learning curve theory.

[25] J. Siems et al. Nas-bench-301 and the case for surrogate benchmarks for neural architecture search. *arXiv preprint arXiv:2008.09777*, 2020.

[26] A. Krizhevsky et al. Learning multiple layers of features from tiny images. 2009.

[27] B. Gu et al. Modelling classification performance for large data sets.

[28] F. Mohr et al. LCDB 1.0: An extensive learning curves database for classification tasks.

[29] HP. Gavin. The levenberg-marquardt algorithm for nonlinear least squares curve-fitting problems.

[30] J. Bradbury et al. JAX: composable transformations of Python+NumPy programs.

[31] J. Bergstra et al. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures.

[32] F. Hutter et al. Algorithm runtime prediction: Methods & evaluation.

[33] F. Pedregosa et al. Scikit-learn: Machine learning in Python. *JMLR*, 12:2825–2830, 2011.

A Visualization of Learning Curves from HPOBench

In this section, we display in Figure 4 the 200 randomly picked learning curves for the four benchmarks considered. It is interesting to observe these curves to notice their diversity. Also, many outliers are visible which makes regular and well behaving learning curves harder to distinguish (and impossible in a linear scale). Then, we present 1,000 randomly selected learning curves colored by their final ranking in Figure 5 and the corresponding heatmaps in Figure 6. In such visualization, we can identify clearly two groups among learning curves (good and bad candidates). Also we can notice that it is possible to identify good candidates only based on the first epoch.

B Visualization of Learning Curves from LCBench

Similarly to what is done on HPOBench in the previous section we perform the same visualization now on LCBench [18]. We present 1,000 randomly selected learning curves colored by their final ranking in Figure 7,8 and the corresponding heatmaps in Figure 9,10. In such visualization, we can identify clearly two groups among learning curves (good and bad candidates). Also we can notice that it is possible to identify good candidates only based on the first epoch.

C Visualization of Learning Curves from YAHPO

Similarly to what is done on LCBench in the previous section we perform the same visualization now on YAHPO [22]. We present 1,000 randomly selected learning curves, colored by their final ranking in Figure 11,12 and the corresponding heatmaps in Figure 13,14. In such visualization, we can identify clearly two groups among learning curves (good and bad candidates). Also we can notice that it is possible to identify good candidates only based on the first epoch. From this results, it is clear that the 1-Epoch baseline would be performing similarly well on a majority of these problems. Now, LCBench only considers tabular datasets. Therefore, within YAHPO we look at NB301 [25] based on Cifar-10 (computer vision dataset) from [26], the only scenario (other than the LCBench scenario) using epochs as the fidelity. We present the results in Fig. 15. From the heatmap, it is clear that the best configuration is already dominating bad configurations from the first epoch. Therefore, 1-Epoch would also perform well on this benchmark.

D Visualization of Learning Curves from JAHS-BENCH

Similarly to was done on HPOBench, LCBench and YAHPO in the previous section we perform the diagnostic visualization now on JAHS-Bench-201 [21]. We present 1,000 randomly selected learning curves as done before, colored by their final ranking in Figure 16 and the corresponding heatmaps in Figure 17. Same conclusions as previous benchmarks can be derived. The best curves are dominant over the red curves. This can be seen well in the top row of the heatmaps Figure 17 which remain blue from the start to the end.

E Experiments on HPOBench

In this section, we present the four problems of the HPOBench benchmark in Figure 18 to complement the Naval Propulsion problem presented in the main part of our study. As it can be seen, all problems have similar outcomes.

F Experiments on LCBench

In this section, we compare the performance of our 1-Epoch baseline on the LCBench [18] benchmark. The experimental setting is similar to the experiments on HPOBench. A random sampling is used for the outer-loop optimization. The minimum fidelity on LCbench is 1 epoch of training, and the maximum is 50 epochs of training. The outer-loop is executed for 200 iterations. The Top-K selection is run with $k = 3$ for 1-Epoch. Table 1 summarizes the results of final test objective (in this case cross-entropy) as well as the final speed up in term of consumed training epochs. To be transpartent about the behaviour of the algorith we also display the search trajectories in Figure 19 and 20. As it can be observed, the test objectives found are close in general dispite a large difference in speed-up. Therefore, if the main bottleneck to execute hyperparameter optimization is compute, using 1-Epoch can be justified.

G Details About HPOBench Search Space

The search space for the 4 tabular benchmarks we used from HPOBench is given by [17] and detailed in table 2. The backbone is a two-layers fully connected neural network with a linear output layer.

H Rober: Robust Bayesian Early Rejection

In this section, we present shortly our improved version of learning curve extrapolation, we call it robust Bayesian early rejection (RoBER). The detailed algorithm of RoBER is provided in Algorithm 1. RoBER closely follows the proposed weighted probabilistic model (WPM) from [8]. WPM consists in estimating the posterior distribution $P(\Theta|C)$ of a mixture distribution of parametric

Dataset	1-Epoch	SHA	50-Epochs	S(1-Epochs)	S(SHA)
APSFailure	0.34 ± 0.00	0.33 ± 0.00	0.33 ± 0.00	28.71	12.29
Amazon employee access	0.59 ± 0.00	0.58 ± 0.00	0.58 ± 0.00	28.71	11.92
Australian	0.48 ± 0.00	0.48 ± 0.00	0.47 ± 0.00	28.71	12.23
Fashion MNIST	1.61 ± 0.01	1.60 ± 0.00	1.60 ± 0.00	28.71	11.10
KDDCup09 appetency	0.41 ± 0.01	0.40 ± 0.01	0.37 ± 0.01	28.71	11.77
MiniBooNE	0.44 ± 0.01	0.43 ± 0.00	0.43 ± 0.00	28.71	11.30
adult	0.51 ± 0.00	0.50 ± 0.00	0.50 ± 0.00	28.71	11.99
airlines	0.66 ± 0.00	0.66 ± 0.00	0.66 ± 0.00	28.71	11.27
albert	0.64 ± 0.00	0.64 ± 0.00	0.64 ± 0.00	28.71	11.59
bank marketing	0.50 ± 0.00	0.49 ± 0.00	0.49 ± 0.00	28.71	13.31
blood transfusion	0.63 ± 0.00	0.63 ± 0.00	0.63 ± 0.00	28.71	13.12
car	1.01 ± 0.02	0.90 ± 0.02	0.90 ± 0.01	28.71	11.40
christine	0.59 ± 0.00	0.60 ± 0.00	0.59 ± 0.00	28.71	12.84
cnae 9	1.55 ± 0.02	1.54 ± 0.01	1.51 ± 0.01	28.71	11.85
connect 4	0.99 ± 0.03	0.93 ± 0.02	0.90 ± 0.02	28.71	11.66
covertype	1.52 ± 0.01	1.51 ± 0.01	1.50 ± 0.01	28.71	12.13
credit g	0.60 ± 0.00	0.59 ± 0.00	0.59 ± 0.00	28.71	12.28
dionis	5.37 ± 0.08	5.24 ± 0.02	5.24 ± 0.02	28.71	11.06
fabert	1.61 ± 0.01	1.59 ± 0.01	1.59 ± 0.00	28.71	12.54
helena	4.55 ± 0.01	4.50 ± 0.01	4.50 ± 0.01	28.71	11.19
higgs	0.61 ± 0.01	0.61 ± 0.00	0.61 ± 0.00	28.71	11.10
jannis	1.17 ± 0.01	1.14 ± 0.01	1.14 ± 0.01	28.71	11.53
jasmine	0.55 ± 0.00	0.55 ± 0.00	0.54 ± 0.00	28.71	12.01
jungle chess	0.81 ± 0.02	0.77 ± 0.01	0.77 ± 0.01	28.71	12.30
kc1	0.58 ± 0.00	0.57 ± 0.00	0.56 ± 0.00	28.71	11.51
kr vs kp	0.39 ± 0.01	0.36 ± 0.01	0.34 ± 0.00	28.71	11.96
mfeat factors	1.54 ± 0.01	1.52 ± 0.00	1.51 ± 0.00	28.71	11.29
nomao	0.36 ± 0.00	0.36 ± 0.00	0.36 ± 0.00	28.71	12.33
numerai28.6	0.69 ± 0.00	0.69 ± 0.00	0.69 ± 0.00	28.71	12.37
phoneme	0.56 ± 0.01	0.51 ± 0.01	0.51 ± 0.01	28.71	11.75
segment	1.49 ± 0.02	1.43 ± 0.02	1.38 ± 0.01	28.71	11.20
shuttle	1.22 ± 0.01	1.21 ± 0.01	1.19 ± 0.01	28.71	13.23
sylvine	0.44 ± 0.01	0.40 ± 0.00	0.40 ± 0.00	28.71	11.64
vehicle	1.13 ± 0.02	1.07 ± 0.01	1.05 ± 0.01	28.71	11.47
volkert	1.99 ± 0.02	1.95 ± 0.01	1.95 ± 0.01	28.71	11.40

Table 1: Table of results from the LCBench benchmark. The final test cross-entropy (mean ± standard error) are displayed as well as the speed up $S(.)$ for 1-Epoch and SHA with respect to using maximum fidelity (50 training epochs for this benchmark).

learning curves models, where $C = \{(z_1, y_1), ..., (z_i, y_i)\}$ is a partial learning curve of the observed scores and Θ is the concatenation of all the parameters (weights of the mixture and parameters of all learning curves models used in the mixture). Once this posterior distribution is known we can evaluate the probability of the score at the final training step $\hat{y}_{\max}$ to be less than y^* the best score observed so far (including all completed evaluations) which formally corresponds to $p := P(\hat{y}_{\max} < y^* | C; \Theta)$. However, after noticing many instabilities in the optimization process of WPM, in RoBER we cut out the mixture distribution

Hyperparameters	Choices
Initial LR	$\{0.0005, 0.001, 0.005, 0.01, 0.05, 0.1\}$
Batch Size	$\{8, 16, 32, 64\}$
LR Schedule	$\{\text{cosine}, \text{fix}\}$
Activation/Layer 1	$\{\text{relu}, \text{tanh}\}$
Activation/Layer 2	$\{\text{relu}, \text{tanh}\}$
Layer 1 Size	$\{16, 32, 64, 128, 256, 512\}$
Layer 2 Size	$\{16, 32, 64, 128, 256, 512\}$
Dropout/Layer 1	$\{0.0, 0.3, 0.6\}$
Dropout/Layer 2	$\{0.0, 0.3, 0.6\}$

Table 2: Hyperparameter search space for the four tabular benchmarks from HPOBench.

and consider only a single model. More precisely, we keep the *mmf4* (Equation 2) model proposed by [27] and assessed in [28] to be good for extrapolation.

$$f_{\text{mmf4}}(z; \Theta) = \frac{\Theta_0 \Theta_1 + \Theta_2 b(z)^{\Theta_3}}{\Theta_1 + b(z)^{\Theta_3}} \tag{1}$$

Then, after observing at least four steps of the learning curve (number of parameters of the model), we estimate the model parameters Θ through non-linear least squares optimization (l. 17 in Algorithm 1). The continuous optimizer used is the Levenberg-Marquart algorithm [29] to which we can provide the exact Jacobian of the vector of residuals $\nabla_\Theta r(\mathbf{z}, \mathbf{y}; \Theta)$ through automatic differentiation with JAX [30] where r is given by:

$$r(\mathbf{z}, \mathbf{y}; \Theta) = \mathbf{y} - \mathcal{C}(\mathbf{z}; \Theta)$$

which combined with just-in-time compilation (JIT) allows for faster and robust estimation of the Θ parameters (previous works only used approximated gradient which makes it less stable). Now, that we have parameters correctly estimating the mean y_i we are interested in incorporating uncertainty about the future and the noise between training steps. Therefore, we follow a Bayesian approach where we have $P(\Theta|C) \propto P(C|\Theta) \cdot P(\Theta)$. The prior on the parameters is given by $P(\Theta) := \mathcal{N}(\hat{\Theta}, 1)$. The likelihood is given by $P(C|\Theta) := \prod_{i=1}^{n} \mathcal{N}(y_i; \hat{y}_i, \sigma^2)$ where $\sigma^2 \sim \text{Exp}(1)$. Finally, the posterior distribution $P(\Theta|C)$ is not exactly computed but we can generate data (possible learning curves) from it using Monte-Carlo Markov Chain (MCMC) (line 18 in Algorithm 1) a common approach to generate data from an untractable densities. A candidate is then discarded only if $\tau \leq p_{\text{worse}}$ with τ a threshold we are in practice setting quite high ($\tau = 0.9$ in our experiments) to avoid discarding models with performance not certainly worse than the current best. In this sense, the method is quite conservative about model early rejecting because early steps will have high overall uncertainty.

Now that we explained our probabilistic model we give additional details on how we handle earliest steps where $z < |\Theta|$. Indeed, for these steps no least-square

estimate is available has we have more degrees of freedom than the number of collected observations. We could remain in an over-parameterized regime and adopt a different optimization approach to alleviate this issue. Instead, we decide to place some safeguard to help us detect specific cases which are stagnation of the learning curve and unexpectedly bad candidates (commonly named "outliers" in data-science). The stagnation in learning curve is managed through a classic early stopping strategy (line 7 in Algorithm 1). Then, the outliers are detected through the simple inter-quantile range methodology, visually we look at the distribution of performances at a given step and consider them outlier when having a lower value than the lower whisker of a box-plot (lines 12-14 in Algorithm 1). It corresponds to discarding models with a score less than $Q_1 - 1.5(Q_3 - Q_1)$ where Q_1, Q_3 are respectively the first and third quartiles of the distribution of observed scores at the given step

$$f_{\text{mmf4}}(z;\Theta) = \frac{\Theta_0\Theta_1 + \Theta_2 b(z)^{\Theta_3}}{\Theta_1 + b(z)^{\Theta_3}} \tag{2}$$

Algorithm 1: Robust Bayesian Early Rejection (RoBER)

1 **Function** `evaluate_with_RoBER(`*config*,τ, z_{max}`)` **is**
2 $C \leftarrow [\,]$;
3 **for** $z \in [|1, z_{max}|]$ **do**
4 `score` $\leftarrow$ Execute training step for `config` ;
5 Append `score` to observations C ;
6
/* Decide if we continue the training */
7 **if** $z_{max} \leq z$ ***or*** *not improved for* $n_{patience}$ *steps* **then**
8 Exit the loop ;
9 $y^* \leftarrow$ Return the maximum objective among all evaluated configurations;
10 $z_{\text{test}} \leftarrow$ Return step at which objective is checked ;
11 **if** $z_{test} = z$ **then**
12 $Q_1 \leftarrow$ 25th percentile of objectives at step z;
13 $Q_3 \leftarrow$ 75th percentile of objectives at step z;
14 **if** $z \leq \min(z_{min}, n_{parameters})$ ***and*** $y_z < Q_1 - 1.5 \cdot (Q_3 - Q_1)$ **then**
15 Exit the loop ;
16 **else**
17 $\hat{\Theta} \leftarrow$ Estimate model parameters with non-linear least-squares ;
18 $p \leftarrow$ Estimate probability of being worse $P(\hat{y}_{\max} < y^* | C; \hat{\Theta})$;
19 **if** $\tau \leq p$ **then**
20 Exit the loop ;
21 **end**
22 **return** last element in C ;
23 **end**

I Experiments with RoBER

In this section, we show that RoBER is more robust than previously proposed learning curve extrapolation methods. The results are presented in Figure 21. It can be seen that RoBER is consistent on the four problems while WPM [8] under-perform on two of the problems 21a and 21d. In addition, experimentally we RoBER is much faster to query compared to WPM which makes it more usable.

J More Details About our Bayesian Optimization Policy

In our experiment, we also combined Bayesian optimization at the outer-optimization level with multi-fidelity agents at the inner-optimizer level. For this we use a Random-Forest surrogate model unlike very frequent Gaussian-Process based approaches or Tree-Parzen Estimation (TPE) [1, 31]. Our Random-Forest is slightly modified as suggested in [32] to use the "best random" split instead of the "best" split which improves significantly the estimation of epistemic uncertainty of the model (based on the Scikit-Learn [33] implementation). Also, the objective is mapped to $[0, 1]$ with a min-max standardization before applying a $\log(y_{\text{min-max}} + \epsilon)$ transformation on it. This equalize the errors on large and small objective values as the average error is being minimized by the learning algorithm. Then, we use the upper confidence-bound (UCB) acquisition function $\alpha(x) = \mu(x) + \kappa \cdot \sigma(x)$ with a cyclic exponential decay to manage the evolution of the exploration-exploitation parameter κ. This scheduler is able to repetitively perform exploration while having moments of strong exploitation (with a *kappa* value cycling to converge to a value close to 0). We adopt this scheme after noticing the empirical efficiency of the TPE algorithm on HPO which has a strong exploitation scheme embedded in it.

K The Problem of Generalization in HPO

In this section, we present the difference between the evolution of the validation error used by the optimizer during hyperparameter optimization and the generalization error (real target). As it can be observed in Figure 22 looking at the evolution of the validation error can be miss-leading of the real generalization error.

L Code

The code is made publicly available with related documentation as part of an existing Hyperameter optimization Python package. Our experiments are organized as followed:

- Benchmark definitions: github.com/deephyper/benchmark

- Library containing the core software and algorithms: github.com/deephyper/deephyper
- Organization, execution and analysis of the experiments of the paper: github.com/deephyper/scalable-bo

The experiments for this paper are located in the `scalable-bo/experiments/local/dhb/` folder of the `deephyper/scalable-bo` repository. The main entry point for the execution is `scalable-bo/src/scalbo/scalbo/search/dbo.py`. Plots and analysis are all located in the `scalable-bo/notebooks/learning-curves/` folder.

(a) Naval Propulsion

(b) Parkinsons Telemonitoring

(c) Protein Structure

(d) Slice Localization

Fig. 4: A set of 200 randomly sampled curves from the different benchmarks in HPOBench. The mean (plain-line) and standard error of the validation objective (RMSE) for each curve are displayed as different random-seeds are provided in HPOBench.

(a) Naval Propulsion

(b) Parkinsons Telemonitoring

(c) Protein Structure

(d) Slice Localization

Fig. 5: Learning curves of 1000 randomly sampled models from HPOBench colored by the rank at maximum fidelity.

(a) NavalPropulsion

(b) ParkinsonsTelemonitoring

(c) ProteinStructure

(d) SliceLocalization

Fig. 6: Evolution of the ranking from minimal to maximal fidelity on 1,000 randomly sampled models from the search space of HPOBench.

(a) APSFailure (b) Amazon employee access (c) Australian (d) Fashion MNIST

(e) KDDCup09 appetency (f) MiniBooNE (g) adult (h) airlines

(i) albert (j) bank marketing (k) blood transfusion (l) car

(m) christine (n) cnae 9 (o) connect 4 (p) covertype

(q) credit g (r) dionis (s) fabert (t) helena

(u) higgs (v) jannis (w) jasmine (x) jungle chess 2pcs raw endgame complete

(y) kc1 (z) kr vs kp

Fig. 7: Learning curves of 1000 randomly sampled models from HPOBench colored by the rank at maximum fidelity (**First Part**).

(a) mfeat factors (b) nomao (c) numerai28.6 (d) phoneme

(e) segment (f) shuttle (g) sylvine (h) vehicle

(i) volkert

Fig. 8: Learning curves of 1000 randomly sampled models from LCBench colored by the rank at maximum fidelity (**Second Part**).

(a) APSFailure

(b) Amazon employee access

(c) Australian

(d) Fashion MNIST

(e) KDDCup09 appetency

(f) MiniBooNE

(g) adult

(h) airlines

(i) albert

(j) bank marketing

(k) blood transfusion

(l) car

(m) christine

(n) cnae 9

(o) connect 4

(p) covertype

(q) credit g

(r) dionis

(s) fabert

(t) helena

(u) higgs

(v) jannis

(w) jasmine

(x) jungle chess 2pcs raw endgame complete

(y) kc1

(z) kr vs kp

Fig. 9: Evolution of the ranking from minimal to maximal fidelity on 1,000 randomly sampled models from the search space of LCBench (**First Part**).

(a) mfeat factors (b) nomao (c) numerai28.6 (d) phoneme

(e) segment (f) shuttle (g) sylvine (h) vehicle

(i) volkert

Fig. 10: Evolution of the ranking from minimal to maximal fidelity on 1,000 randomly sampled models from the search space of LCBench (**Second Part**).

(a) APSFailure (b) Amazon employee access (c) Australian (d) Fashion MNIST

(e) KDDCup09 appetency (f) MiniBooNE (g) adult (h) airlines

(i) albert (j) bank marketing (k) blood transfusion (l) car

(m) christine (n) cnae 9 (o) connect 4 (p) covertype

(q) credit g (r) dionis (s) fabert (t) helena

(u) higgs (v) jannis (w) jasmine (x) jungle chess 2pcs raw endgame complete

(y) kc1 (z) kr vs kp

Fig. 11: Learning curves of the 1000 randomly sampled models from LCBench now in YAHPO colored by the rank at maximum fidelity (**First Part**).

(a) mfeat factors (b) nomao (c) numerai28.6 (d) phoneme

(e) segment (f) shuttle (g) sylvine (h) vehicle

(i) volkert

Fig. 12: Learning curves of the 1000 randomly sampled models from LCBench now in YAHPO colored by the rank at maximum fidelity (**Second Part**).

(a) APSFailure

(b) Amazon employee access

(c) Australian

(d) Fashion MNIST

(e) KDDCup09 appetency

(f) MiniBooNE

(g) adult

(h) airlines

(i) albert

(j) bank marketing

(k) blood transfusion

(l) car

(m) christine

(n) cnae 9

(o) connect 4

(p) covertype

(q) credit g

(r) dionis

(s) fabert

(t) helena

(u) higgs

(v) jannis

(w) jasmine

(x) jungle chess 2pcs raw endgame complete

(y) kc1

(z) kr vs kp

Fig. 13: Evolution of the ranking from minimal to maximal fidelity on 1,000 randomly sampled models from the search space of YAHPO (**First Part**).

(a) mfeat factors (b) nomao (c) numerai28.6 (d) phoneme

(e) segment (f) shuttle (g) sylvine (h) vehicle

(i) volkert

Fig. 14: Evolution of the ranking from minimal to maximal fidelity on 1,000 randomly sampled models from the search space of YAHPO (**Second Part**).

(a) NB301 from YAHPO (b) NB301 from YAHPO

Fig. 15: (Left) Learning curves colored by the rank at maximum fidelity (98 epochs) and (Right) evolution of the ranking from minimal to maximal fidelity both on 1,000 randomly sampled models from the NB301 benchmark on the Cifar10 dataset from YAHPO.

(a) Cifar10 (b) Colorectal Histology (c) Fashion-MNIST

Fig. 16: Learning curves colored by the rank at maximum fidelity from JAHS-Bench-201.

(a) Cifar10 (b) Colorectal Histology (c) Fashion-MNIST

Fig. 17: Evolution of the ranking from minimal to maximal fidelity on 1,000 randomly sampled models from JAHS-Bench-201.

Fig. 18: Comparing the performance (mean and standard error) of various early discarding strategies (all combined with with random search) on the HPOBench benchmark: 100-Epochs and 1-Epoch use a fixed number of epoch while Successive Halving (SHA), Hyperband (HB) and Learning Curve Extrapolation (LCE) manage the number of epochs dynamically. Using as stopping criterion 200 iterations of the search agent, dynamic strategies consume fewer training epochs than 100-Epochs, but, 1-Epoch performs yet better than all methods. In particular, it consumes 40 times fewer epochs than 100-Epochs.

(a) APSFailure

(b) Amazon employee access

(c) Australian

(d) Fashion MNIST

(e) KDDCup09 appetency

(f) MiniBooNE

(g) adult

(h) airlines

(i) albert

(j) bank marketing

(k) blood transfusion

(l) car

(m) christine

(n) cnae 9

(o) connect 4

(p) covertype

(q) credit g

(r) dionis

(s) fabert

(t) helena

(u) higgs

(v) jannis

(w) jasmine

(x) jungle chess 2pcs raw endgame complete

(y) kc1

(z) kr vs kp

Fig. 19: Comparing pruners 1-Epoch, 50-Epochs and SHA on the LCBench benchmark (**First Part**).

(a) mfeat factors (b) nomao (c) numerai28.6 (d) phoneme

(e) segment (f) shuttle (g) sylvine (h) vehicle

(i) volkert

Fig. 20: Comparing pruners 1-Epoch, 50-Epochs and SHA on the LCBench benchmark (**Second Part**).

(a) Naval Propulsion (b) Parkinsons Telemonitoring

(c) Protein Structure (d) Slice Localization

Fig. 21: Comparing RoBER with Weighted Probabilistic Model (WPM) on HPOBench. RoBER is consistently better on all the problems and the WPM is something very unstable and worse than the basic 100-Epochs evaluation strategy.

(a) Naval Propulsion

(b) Parkinsons Telemonitoring

(c) Protein Structure

(d) Slice Localization

Fig. 22: Looking at the Generalization Error in HPO for the 1-Epoch multi-fidelity agent.