

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

Romain Egele^{1,3}, Isabelle Guyon^{1,2}, Yixuan Sun³, Prasanna Balaprakash⁴

1- LISN, U. Paris-Saclay, France; 2- Google, USA & ChaLearn, USA;
3- MCS, Argonne Ntl Lab, USA; 4- CCSD, Oak Ridge Ntl Lab, USA

Abstract.

Hyperparameter optimization (HPO) is crucial for fine-tuning machine learning models, but it can be computationally expensive. To reduce costs, Multi-fidelity HPO (MF-HPO) leverages intermediate accuracy levels in the learning process and discards low-performing models early on. We conducted a comparison of various representative MF-HPO methods against a simple baseline on classical benchmark data. The baseline involved discarding all models except the Top- K after training for only one epoch, followed by further training to select the best model. Surprisingly, this baseline achieved similar results to its counterparts, while requiring an order of magnitude less computation. Upon analyzing the learning curves of the benchmark data, we observed a few dominant learning curves, which explained the success of our baseline. This suggests that researchers should (1) always use the suggested baseline in benchmarks and (2) broaden the diversity of MF-HPO benchmarks to include more complex cases.

1 Introduction

Hyperparameter optimization (HPO) is a key component of AutoML systems. It aims to find the best configuration of a machine learning (ML) pipeline, which consists of data and model components. Hyperparameters (HP) are parameters that cannot be learned directly with the primary learning algorithm (e.g., gradient descent), but affect the learning process and the performance of the pipeline. Those may include number of layers and units per layer in deep networks, learning rates, etc. HPO is usually formulated as a black-box optimization problem[1], where a function maps a HP configuration to a performance score. However, evaluating this function can be costly and time-consuming. Therefore, Multi-fidelity HPO (MF-HPO) methods have been proposed [2, 3], which use intermediate learning machine performance estimates to reduce the computational cost and speed up the optimization process. In multi-fidelity algorithms, fidelity refers to the level of accuracy or resolution at which a machine learning model is trained and evaluated. Different levels of fidelity can be used to balance the trade-off between computational cost and accuracy. For example, a low fidelity level could involve training and evaluating a model on a small subset of the data, while a high fidelity level would involve using the entire dataset. Multi-fidelity algorithms use this concept to optimize hyperparameters and other model parameters efficiently by exploring the model space using a combination of different fidelity levels. The accuracy obtained at each fidelity level is used to guide

the optimization process towards better performing models while minimizing the computational cost.

Fig. 1 presents the generic components of a multi-fidelity HPO. The input to HPO is the HP search space. The output is a trained model with the corresponding hyperparameters. HPO is a bi-level optimization problem: the upper level (**outer loop**) optimizes the HP of the ML pipeline while the lower level (**inner loop**) optimizes the parameters of the learning machine, given the HP configuration of the upper level. After completing the search process, a final step of **model selection** is performed before returning the trained model. In regular HPO, the inner loop is halted by a given stopping criterion. A natural improvement is to consider the multi-fidelity setting, where intermediate performance estimates are obtained at different stages of the inner optimization process (as a function of the number of training examples or training epochs). This involves monitoring a "learning curve."

This study evaluates the effectiveness of early discarding policies in the **inner loop**, while random search is used for the **outer loop**. We use widely-used approaches for early discarding, performing denovo search without prior knowledge of learning curve typology or meta-learning involvement.

We consider two baselines at each extreme. The first evaluates each HP configuration candidate at maximum fidelity, i.e., after being fully trained within the overall time budget (100-Epochs). The second evaluates them at minimum fidelity, after only one epoch of training. We demonstrate the effectiveness of the 1-Epoch approach and explain why it is successful. Our findings provide a new perspective on previous studies that omitted this baseline. Only one study [4] included a similar baseline and reported results similar to ours. Therefore, we advocate for including the 1-Epoch baseline in further MF-HPO benchmarks.

2 Related Work

Our study focuses on methods, which train only a single model at a time, but keep all check-points for further reference. Early discarding means switching to training a model with another HP configuration, before attaining the maximum number of epochs allowed. A typical example of such strategy, some-

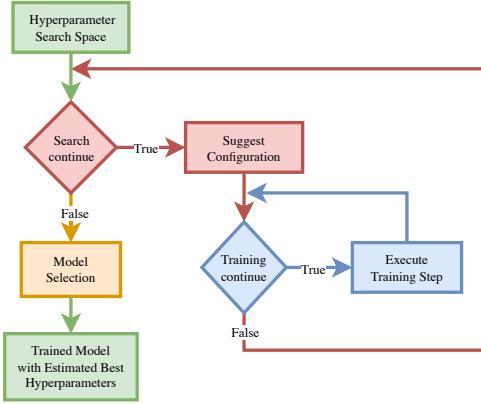


Fig. 1: Hyperparameter optimization and its components including **input/output**, **outer optimization loop exploring new HP configurations**, **inner optimization loop incrementally allocating new training "steps"** and **model selection**.

times referred to as “vertical” model selection [5] is Asynchronous Successive Halving [6] (SHA). Several methods can be adapted to this setting, including Hyperband [3], which can explore different levels of fidelity to differentiate noisy learning curves; Learning Curve Extrapolation [7] (LCE), which can observe early performances and extrapolate future performances to decide whether training should continue; FABOLAS [8], which learns correlations in the candidates’ ranking between different levels of fidelity; Bayesian Optimization Hyperband [9], which embeds Bayesian optimization in Hyperband to sample candidates more efficiently; Learning Curve with Support Vector Regression [10], which predicts final performance based on the configuration and early observations; Learning Curve with Bayesian neural network [11], which is similar to the previous method but switches the model with a Bayesian neural network; and Trace Aware Knowledge-Gradient [12], which leverages an observed curve to update the posterior distribution of a Gaussian process more efficiently.

The previous works had some limitations, which were primarily due to the assumptions made about the learning curve. For example, methods based on SHA assume that the discarded learning curves will not cross in the future, since only the Top- K models are allowed to continue at any given step. In this context, models that start slowly are often discarded. This phenomenon is known as “short-horizon bias” [13]. For methods based on LCE, they either assume a “biased” parametric model, or they are more generic but require a more expensive initialization[14]. The second limitation is that the configuration returned is often the lowest validation error observed during the search. However, this approach is quite limiting because it does not consider more sophisticated model selection schemes such as cross-validation or Hyperband. Therefore, we added the **model selection** block in Fig. 1.

Benchmarks play a critical role in the design and development of HPO methods. We have surveyed several recent benchmarks for learning curves that are continuously evolving, such as HPOBench [15, 16], LCBench [17], JAHS-Bench-201 [18], and YAHPO-Gym [19].

3 Method

Our experiments include representative state-of-the-art methods in multi-fidelity HPO research. The first method considered is **Successive Halving** (SHA) [2] which we consider in its asynchronous variant [6] to be able to handle a stream of learners. SHA decides to continue the training loop at exponentially increasing steps called rungs (based on learning curves [20] theory). At each rung, it continues training only if the current observed score is among the Top- K scores and discards the model otherwise. This method is error-prone when learning curves are noisy. To make SHA more robust **Hyperband** (HB) [3] enforces the exploration of more levels of fidelity on top of SHA to reduce the impact of noise in decisions. However this is at the expense of consuming more training steps than SHA. Finally, the last approach we consider consists in using a surrogate model to predict the future of the learning curve, called **Learning Curve Extrapolation**

(LCE) [7]. In theory, this approach is less likely to suffer from “short-horizon” bias unless the learning curve surrogate is biased. The method also suffers from instabilities in its original implementation [7]. To address this shortcoming, we are using our own implementation of this method (See the RoBER method described in supplemental material¹).

We compare these methods with two baselines: the maximum fidelity baseline (**100-Epochs**), which trains all methods for 100 Epochs and selects the best one and a minimal fidelity baseline (**1-Epoch**), which comprises in fact two phases: (i) exploration search phase pre-selecting the Top- K after only 1 Epoch (minimum fidelity blue loop in Fig. 1); (ii) model selection phase evaluating the Top- K at maximum fidelity and returning the best model. 1-Epoch is expected to perform poorly if there is a strong “short-horizon” bias (see previous section for a definition) and noisy learning curves.

4 Experimental Results

In our analysis we looked at four benchmarks: HPOBench, LCBench, YAHPGym and JAHS-Bench. But, since they all yielded similar results, for brevity we only expose the Naval Propulsion problem from HPOBench. We provide complementary materials¹ with our full set of results.

To visualize the algorithm progress, we display the test error as a function of training epochs [19] (Fig.1). This type of learning curve weighs training iterations (epochs) equally for all HP configurations, which may be deceiving since they can vary in computational cost. Still, this is a convenient simple method abstracting from

details of implementation. We configured the number of iterations for the outer-loop optimization to 200 (red loop). As a result, the maximum fidelity policy (100-Epochs), consumes the maximum number of 20,000 training epochs. In contrast, 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.

Figure 2 shows the evolution of the test RMSE as a function of the number of training epochs used. The key observation is that the test RMSE for the 1-Epoch (blue curve) and 100-Epochs (orange curve) strategies is similar at the final point. However, the 1-Epoch policy uses 40 times fewer training epochs than its counterpart, the 100-Epoch policy ($20,000/500=40$), as per our experimental setup, which is evident from the last point of the blue and orange

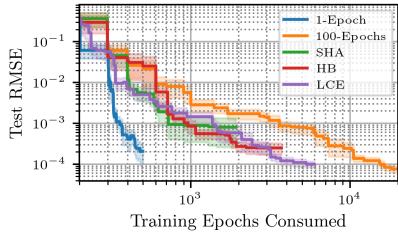
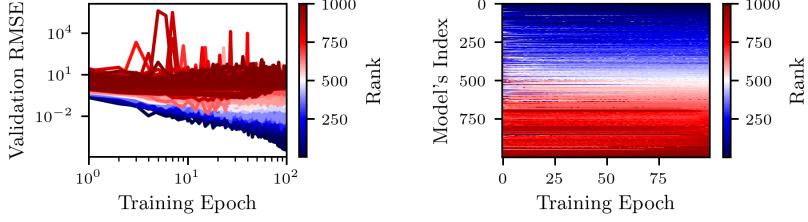


Fig. 2: Comparing the performance (mean and standard error) of various early discarding strategies (all combined with random search) on the Naval Propulsion problem from the HPOBench benchmark.

¹https://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. (a) Models can be selected from the first epoch. (b) The ranking stabilizes faster among the best models.

curves. Additionally, we note that the more complex agents (SHA, HB, and LCE) do not differ significantly from each other and all result in similar final test RMSE, but they consume significantly more epochs. Hyperband consumes slightly more resources than SHA, which is consistent with its design. SHA consumes the least training epochs, about 10 times fewer epochs than 100-Epochs, but still, four times more than 1-Epoch.

After observing the performance of training for only one epoch during the search, we will explain why. In Fig. 3a, we display 1,000 randomly sampled learning curves from the same benchmark, colored by their ranking at maximum fidelity. Low ranks, colored in blue, correspond to good models, while high ranks, colored in red, correspond to bad models. This visualization reveals that the groups of bad and good models can be identified in the first epoch of training, making selection with one epoch effective. Some noise exists between models, as seen in the blue curves, necessitating the top- K tournament selection to differentiate among different models. The noise in the ranking for the same benchmark can be visualized in Fig. 3b, where the ranking for the same models at each training epoch is displayed. The first half (before 50 epochs) has more noise than the second half, and 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

In this study, we investigated the optimization of hyperparameters for machine learning models with minimal training steps using MF-HPO agents. We compared various state-of-the-art multi-fidelity policies and discovered that a new simple baseline method, not considered in prior benchmarks, namely the 1-Epoch strategy assessing candidate configuration at lowest fidelity, performs surprisingly well. This was traced to the fact that good and bad models can be distinguished early in the training process. This phenomenon occurs frequently in benchmarks commonly used in the literature. Therefore, this point to the need to include 1-Epoch in forthcoming benchmarks and eventually designing new harder (possi-

bly more realistic) benchmarks that will defeat it. Our work is limitated to using “epoch” as a unit of fidelity. While this is convenient and appealing to conduct studies independent of hardware implementation considerations, practical application settings may require considering wall time or other options as units of fidelity.

References

- [1] T. Yu et al. Hyper-parameter optimization: A review of algorithms and applications. *arXiv preprint arXiv:2003.05689*, 2020.
- [2] K. Jamieson et al. Non-stochastic best arm identification and hyperparameter optimization. In *Artificial intelligence and statistics*, pages 240–248. PMLR, 2016.
- [3] L. Li et al. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816, 2017.
- [4] O. Bohdal et al. PASHA: Efficient HPO and NAS with progressive resource allocation. In *The Eleventh International Conference on Learning Representations*, 2023.
- [5] F. Mohr et al. Learning curves for decision making in supervised machine learning—a survey. *arXiv preprint arXiv:2201.12150*, 2022.
- [6] L. Li et al. A system for massively parallel hyperparameter tuning. *Proceedings of Machine Learning and Systems*, 2:230–246, 2020.
- [7] T. Domhan et al. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Twenty-fourth international joint conference on artificial intelligence*, 2015.
- [8] A. Klein et al. Fast bayesian optimization of machine learning hyperparameters on large datasets. In *Artificial intelligence and statistics*, pages 528–536. PMLR, 2017.
- [9] S. Falkner et al. Bohb: Robust and efficient hyperparameter optimization at scale. In *International Conference on Machine Learning*, pages 1437–1446. PMLR, 2018.
- [10] B. Baker et al. Accelerating neural architecture search using performance prediction. *arXiv preprint arXiv:1705.10823*, 2017.
- [11] A. Klein et al. Learning curve prediction with bayesian neural networks. In *International Conference on Learning Representations*, 2017.
- [12] J. Wu et al. Practical multi-fidelity bayesian optimization for hyperparameter tuning. In *Uncertainty in Artificial Intelligence*, pages 788–798. PMLR, 2020.
- [13] Y. Wu et al. Understanding short-horizon bias in stochastic meta-optimization. In *International Conference on Learning Representations*, 2018.
- [14] C. White et al. How powerful are performance predictors in neural architecture search? *Advances in Neural Information Processing Systems*, 34:28454–28469, 2021.
- [15] K. Eggensperger et al. Hpobench: A collection of reproducible multi-fidelity benchmark problems for hpo. *arXiv preprint arXiv:2109.06716*, 2021.
- [16] A. Klein et al. Tabular benchmarks for joint architecture and hyperparameter optimization. *arXiv preprint arXiv:1905.04970*, 2019.
- [17] L. Zimmer et al. Auto-pytorch: Multi-fidelity metalearning for efficient and robust autodl. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(9):3079–3090, 2021.
- [18] A. Bansal et al. Jahs-bench-201: A foundation for research on joint architecture and hyperparameter search. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022.
- [19] F. Pfisterer et al. Yahpo gym—an efficient multi-objective multi-fidelity benchmark for hyperparameter optimization. In *International Conference on Automated Machine Learning*, pages 3–1. PMLR, 2022.

- [20] M. Hutter. Learning curve theory. *arXiv preprint arXiv:2102.04074*, 2021.
- [21] B. Gu et al. Modelling classification performance for large data sets. In *Proceedings of the Second International Conference on Advances in Web-Age Information Management, WAIM '01*, page 317–328, Berlin, Heidelberg, 2001. Springer-Verlag.
- [22] HP. Gavin. The levenberg-marquardt algorithm for nonlinear least squares curve-fitting problems. *Department of Civil and Environmental Engineering, Duke University*, 19, 2019.
- [23] J. Bradbury et al. JAX: composable transformations of Python+NumPy programs, 2018.
- [24] J. Bergstra et al. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, 24, 2011.
- [25] J. Bergstra et al. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *International conference on machine learning*, pages 115–123. PMLR, 2013.

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 [?]. 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 [19]. 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 [?] based on Cifar-10 (computer vision dataset) from [?], 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 what was done on HPOBench, LCbench and YAHPO in the previous section we perform the diagnostic visualization now on JAHS-Bench-201 [18]. 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 [?] 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 transparent about the behaviour of the algorithm we also display the search trajectories in Figure 19 and 20. As it can be observed, the test objectives found are close in general despite 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 [16] 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 [7]. 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 $C = \{(z_1, y_1), \dots, (z_i, y_i)\}$ 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	{cosine, fix}
Activation/Layer 1	{relu, tanh}
Activation/Layer 2	{relu, 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 [21] and assessed in [?] to be good for extrapolation.

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

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 [22] 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 [23] 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

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

Algorithm 1: Robust Bayesian Early Rejection (RoBER)

```

1 Function evaluate_with_RoBER(config,  $\tau$ ,  $z_{max}$ ) is
2    $C \leftarrow []$  ;
3   for  $z \in [1, z_{max}]$  do
4      $\text{score} \leftarrow$  Execute training step for config ;
5     Append score to observations  $C$  ;
6
7     /* Decide if we continue the training */  

8     if  $z_{max} \leq z$  or not improved for  $n_{patience}$  steps then  

9       | Exit the loop ;  

10       $y^* \leftarrow$  Return the maximum objective among all evaluated configurations;  

11       $z_{test} \leftarrow$  Return step at which objective is checked ;  

12      if  $z_{test} = z$  then  

13        |  $Q_1 \leftarrow$  25th percentile of objectives at step  $z$ ;  

14        |  $Q_3 \leftarrow$  75th percentile of objectives at step  $z$ ;  

15        | if  $z \leq \min(z_{min}, n_{parameters})$  and  $y_z < Q_1 - 1.5 \cdot (Q_3 - Q_1)$  then  

16          | | Exit the loop ;  

17        | | else  

18          | | |  $\hat{\Theta} \leftarrow$  Estimate model parameters with non-linear least-squares ;  

19          | | |  $p \leftarrow$  Estimate probability of being worse  $P(\hat{y}_{max} < y^* | C; \hat{\Theta})$  ;  

20          | | | if  $\tau \leq p$  then  

21            | | | | Exit the loop ;  

22      end  

23      return last element in  $C$  ;
24 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 [7] 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) [24, 25]. Our Random-Forest is slightly modified as suggested in [?] 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 [?] implementation). Also, the objective is mapped to $[0, 1]$ with a min-max standardization before applying a $\log(y_{\min-\max} + \epsilon)$ 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 Hyperparameter 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.

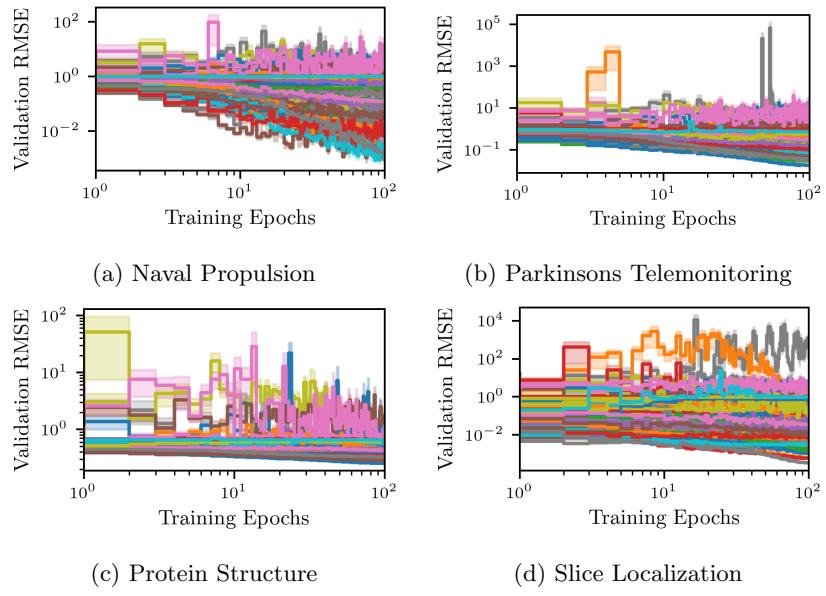


Fig. 4: A set of 200 randomly sampled curves from the different benchmarks in HPOBench. The mean (plain-line) and standard error of the validation objective (RMSE) for each curve are displayed as different random-seeds are provided in HPOBench.

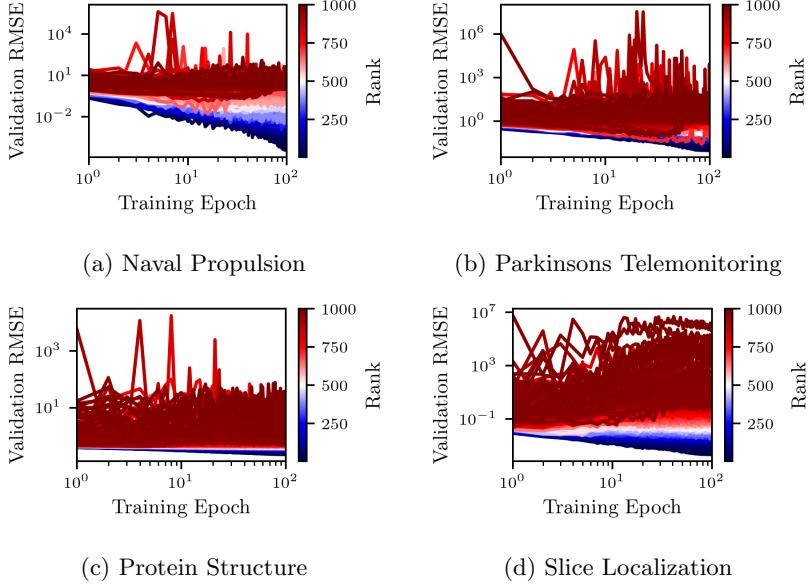


Fig. 5: Learning curves of 1000 randomly sampled models from HPOBench colored by the rank at maximum fidelity.

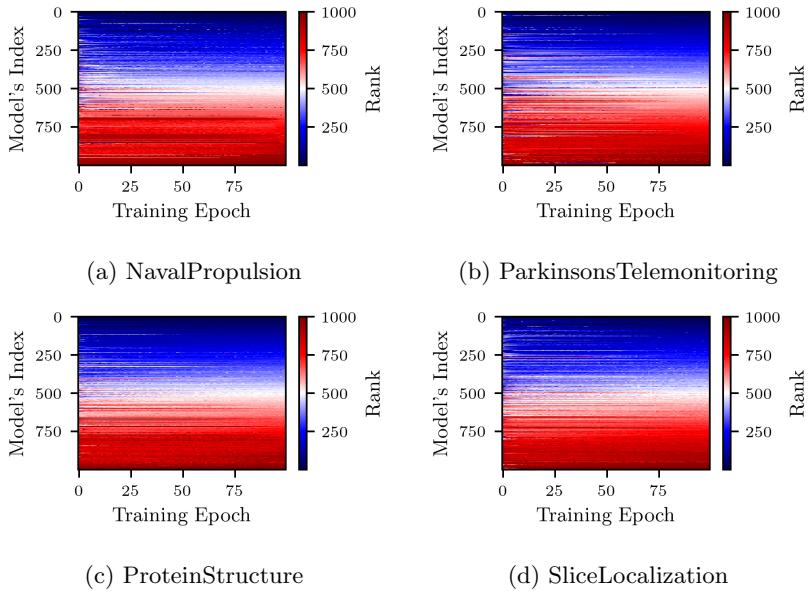


Fig. 6: Evolution of the ranking from minimal to maximal fidelity on 1,000 randomly sampled models from the search space of HPOBench.

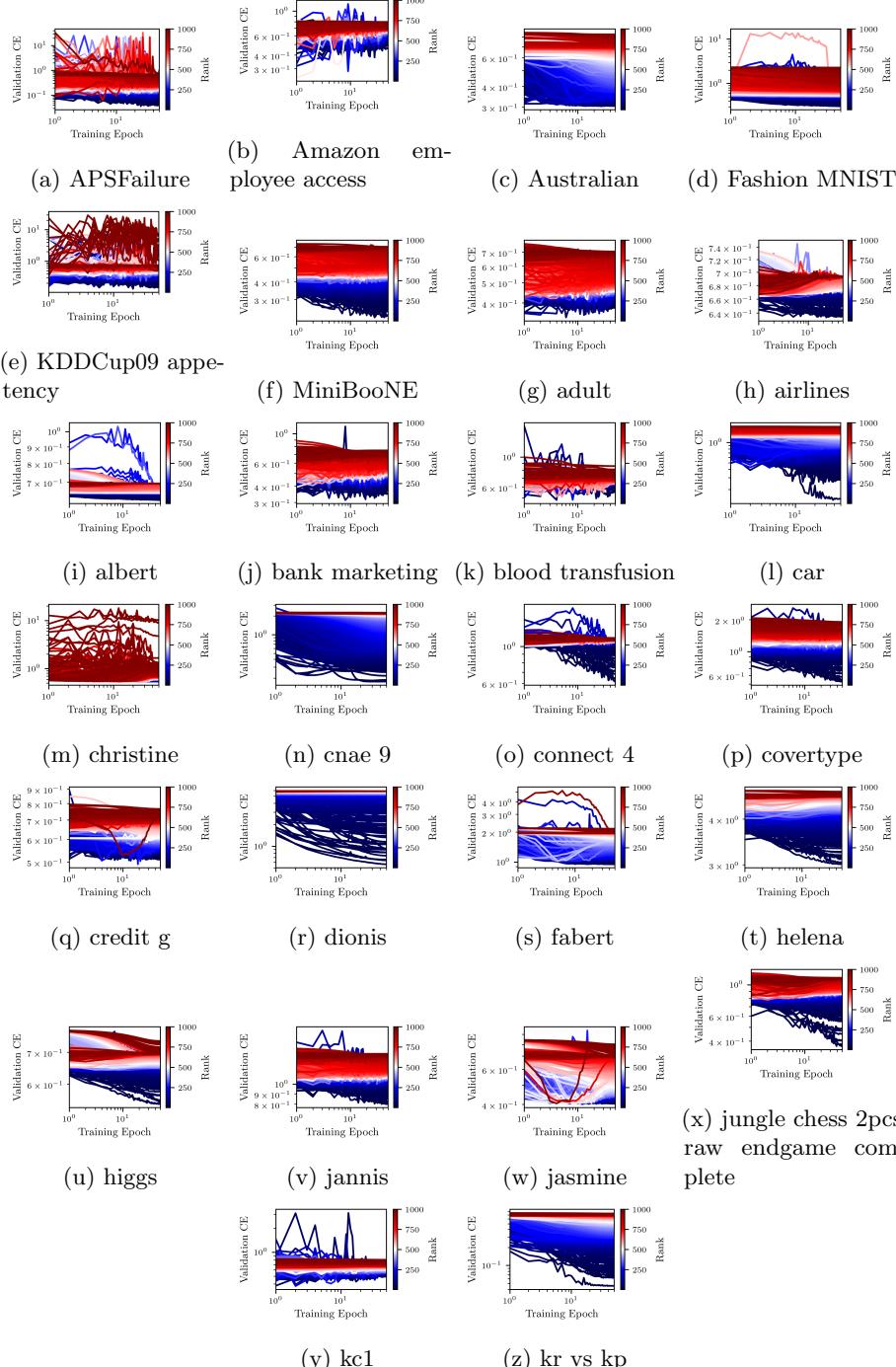


Fig. 7: Learning curves of 1000 randomly sampled models from HPOBench colored by the rank at maximum fidelity (**First Part**).

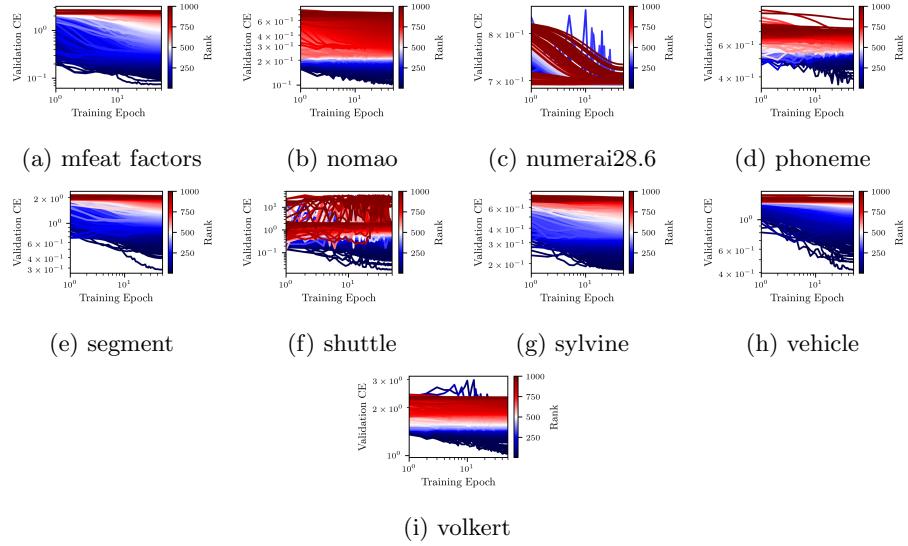


Fig. 8: Learning curves of 1000 randomly sampled models from LCBench colored by the rank at maximum fidelity (**Second Part**).

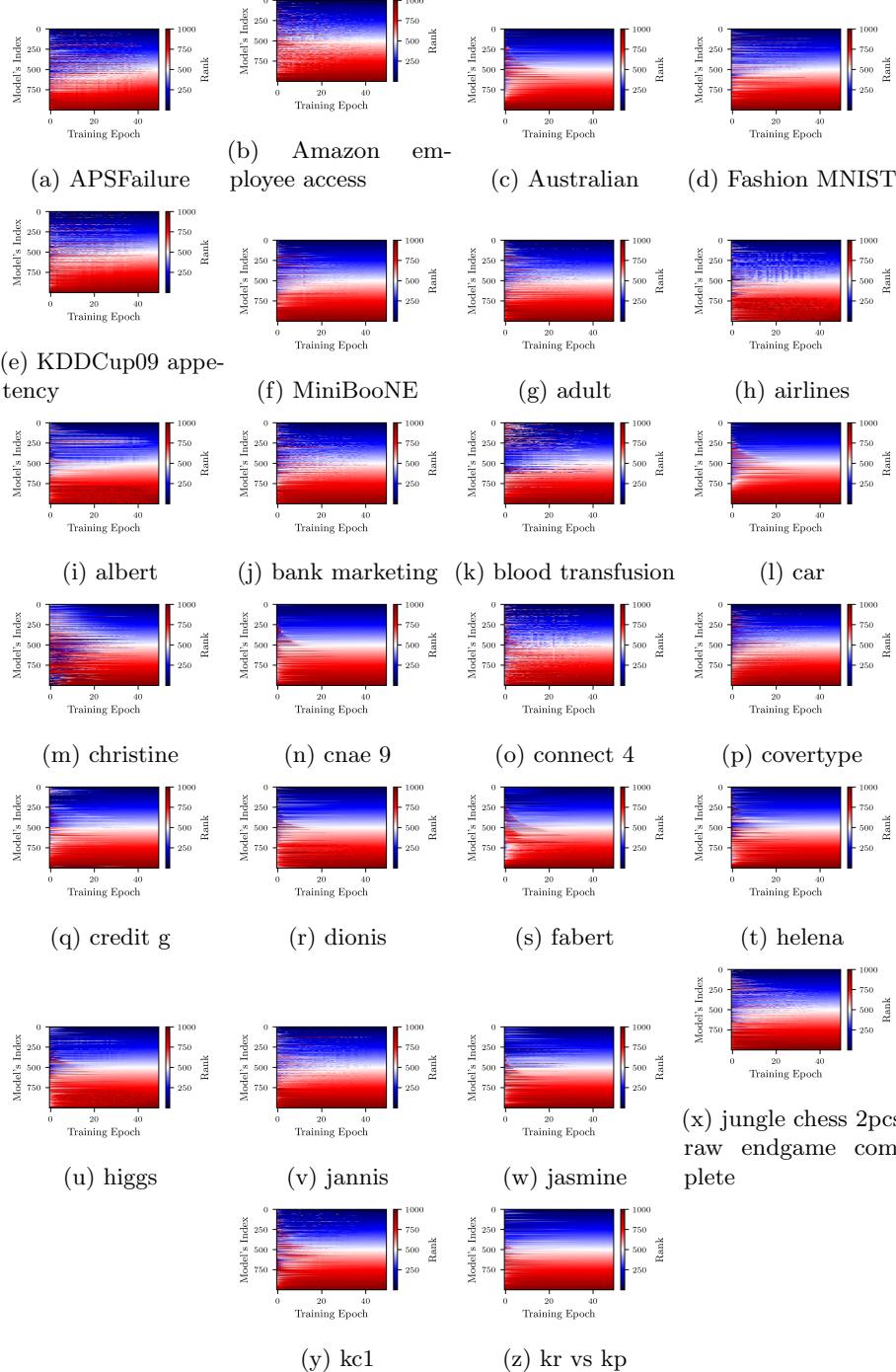


Fig. 9: Evolution of the ranking from minimal to maximal fidelity on 1,000 randomly sampled models from the search space of LCBench (**First Part**).

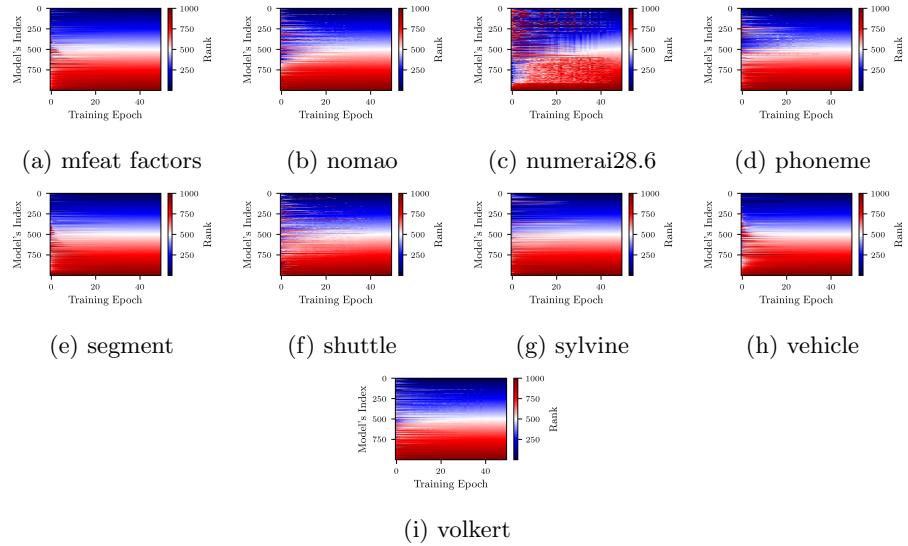


Fig. 10: Evolution of the ranking from minimal to maximal fidelity on 1,000 randomly sampled models from the search space of LCBench (**Second Part**).

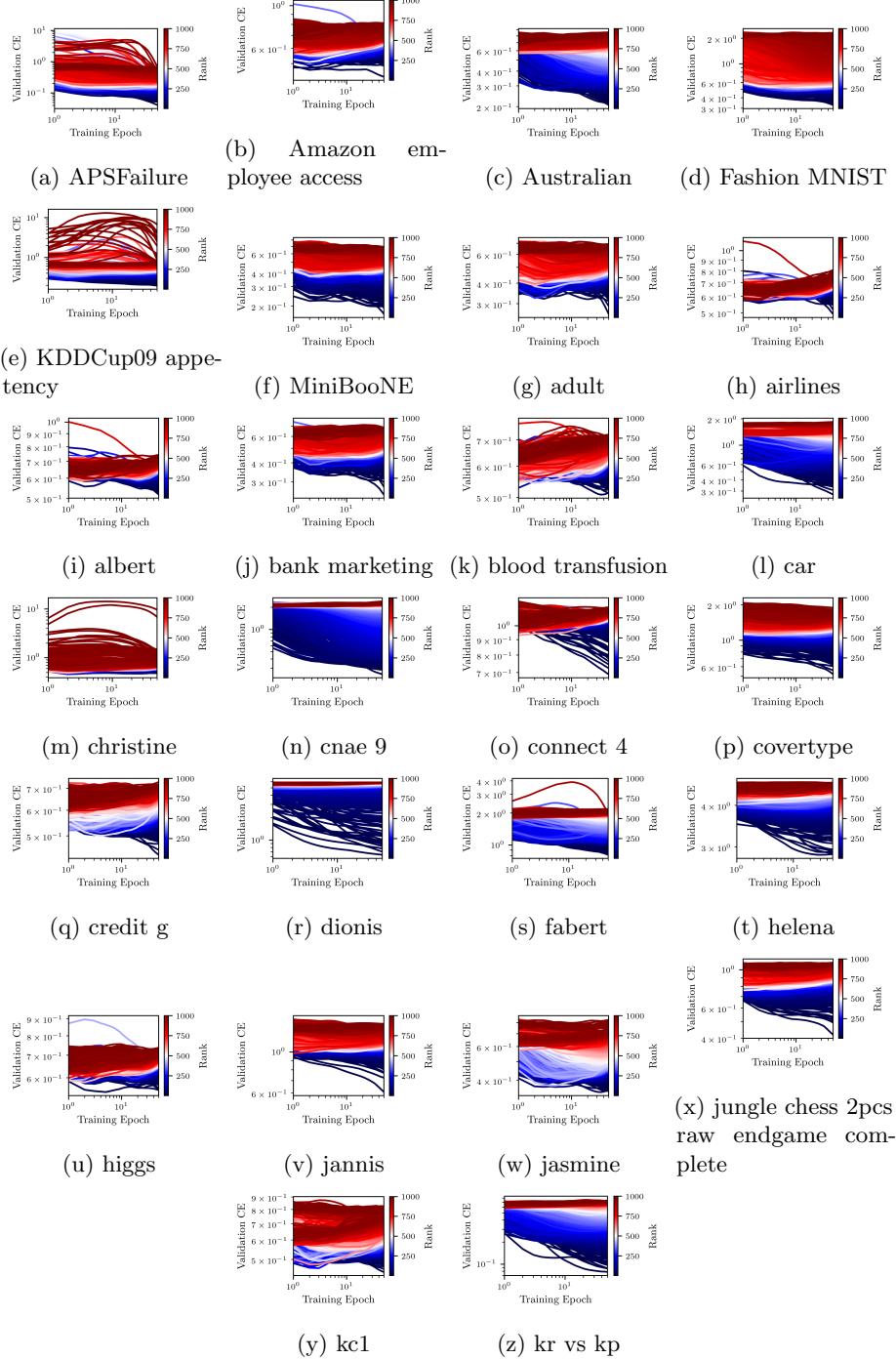


Fig. 11: Learning curves of the 1000 randomly sampled models from LCbench now in YAHPO colored by the rank at maximum fidelity (**First Part**).

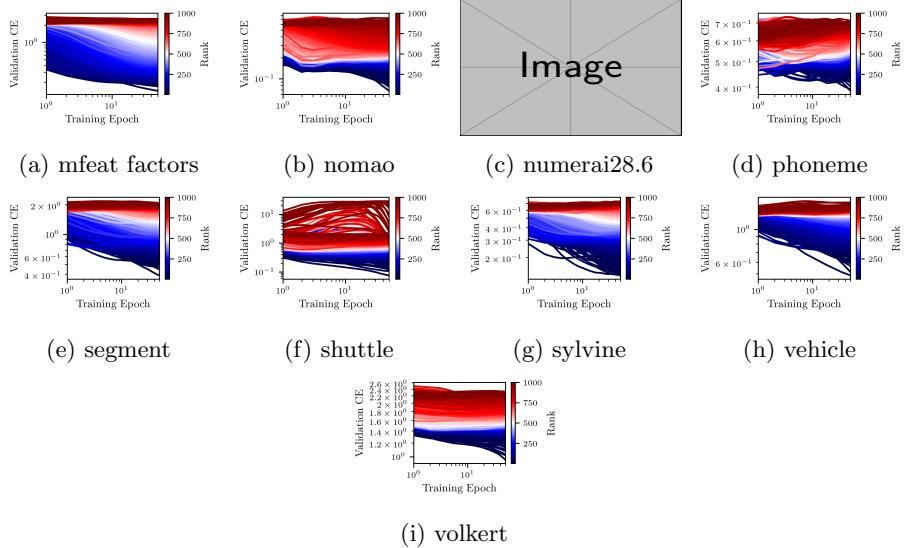


Fig. 12: Learning curves of the 1000 randomly sampled models from LC-Bench now in YAHPO colored by the rank at maximum fidelity (**Second Part**).

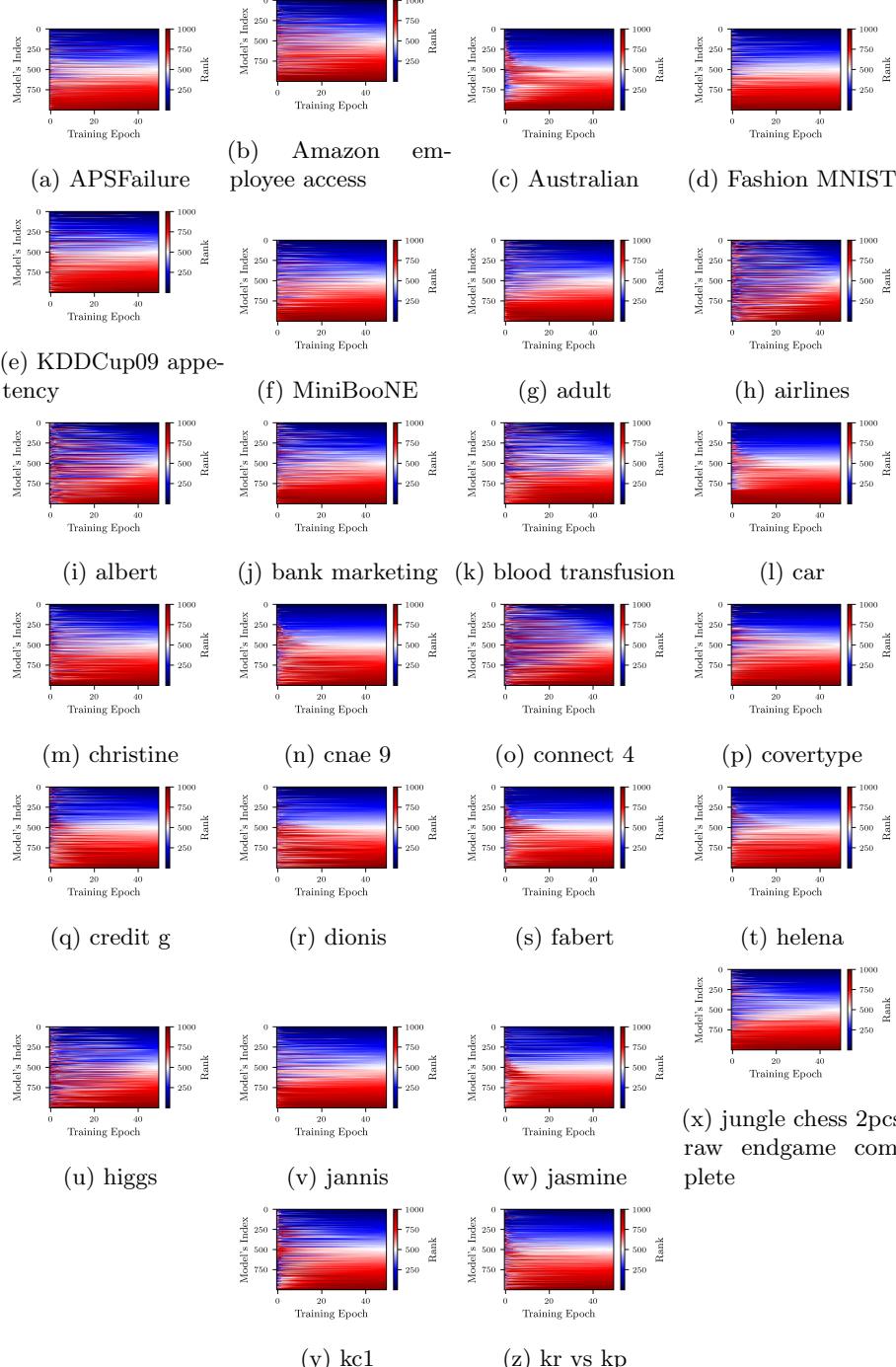


Fig. 13: Evolution of the ranking from minimal to maximal fidelity on 1,000 randomly sampled models from the search space of YAHPO (**First Part**).

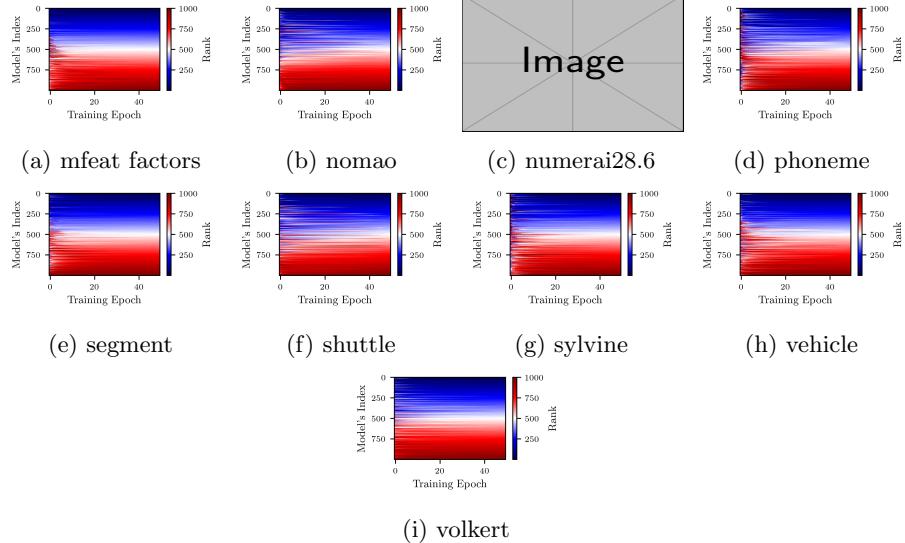


Fig. 14: Evolution of the ranking from minimal to maximal fidelity on 1,000 randomly sampled models from the search space of YAHPO (**Second Part**).

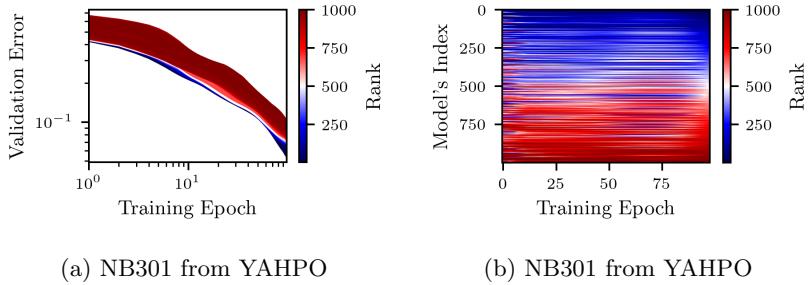


Fig. 15: (Left) Learning curves colored by the rank at maximum fidelity (98 epochs) and (Right) evolution of the ranking from minimal to maximal fidelity both on 1,000 randomly sampled models from the NB301 benchmark on the Cifar10 dataset from YAHPO.

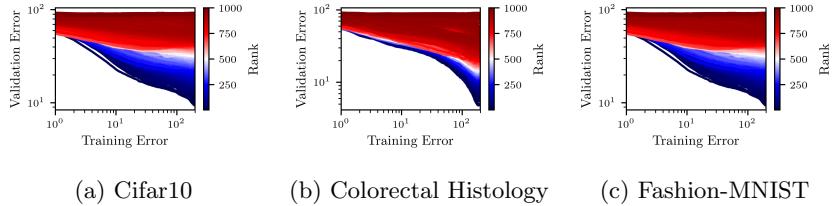
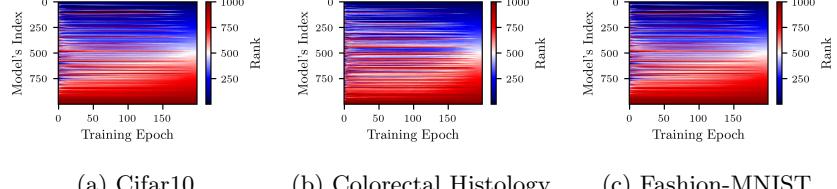


Fig. 16: Learning curves colored by the rank at maximum fidelity from JAHS-Bench-201.



(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.

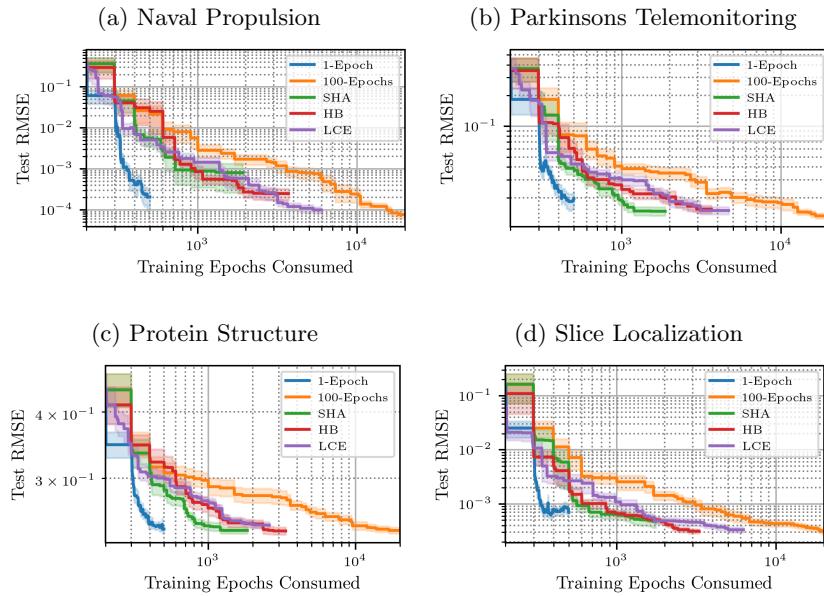


Fig. 18: Comparing the performance (mean and standard error) of various early discarding strategies (all combined with random search) on the HPOBench benchmark: 100-Epochs and 1-Epoch use a fixed number of epoch while Successive Halving (SHA), Hyperband (HB) and Learning Curve Extrapolation (LCE) manage 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.

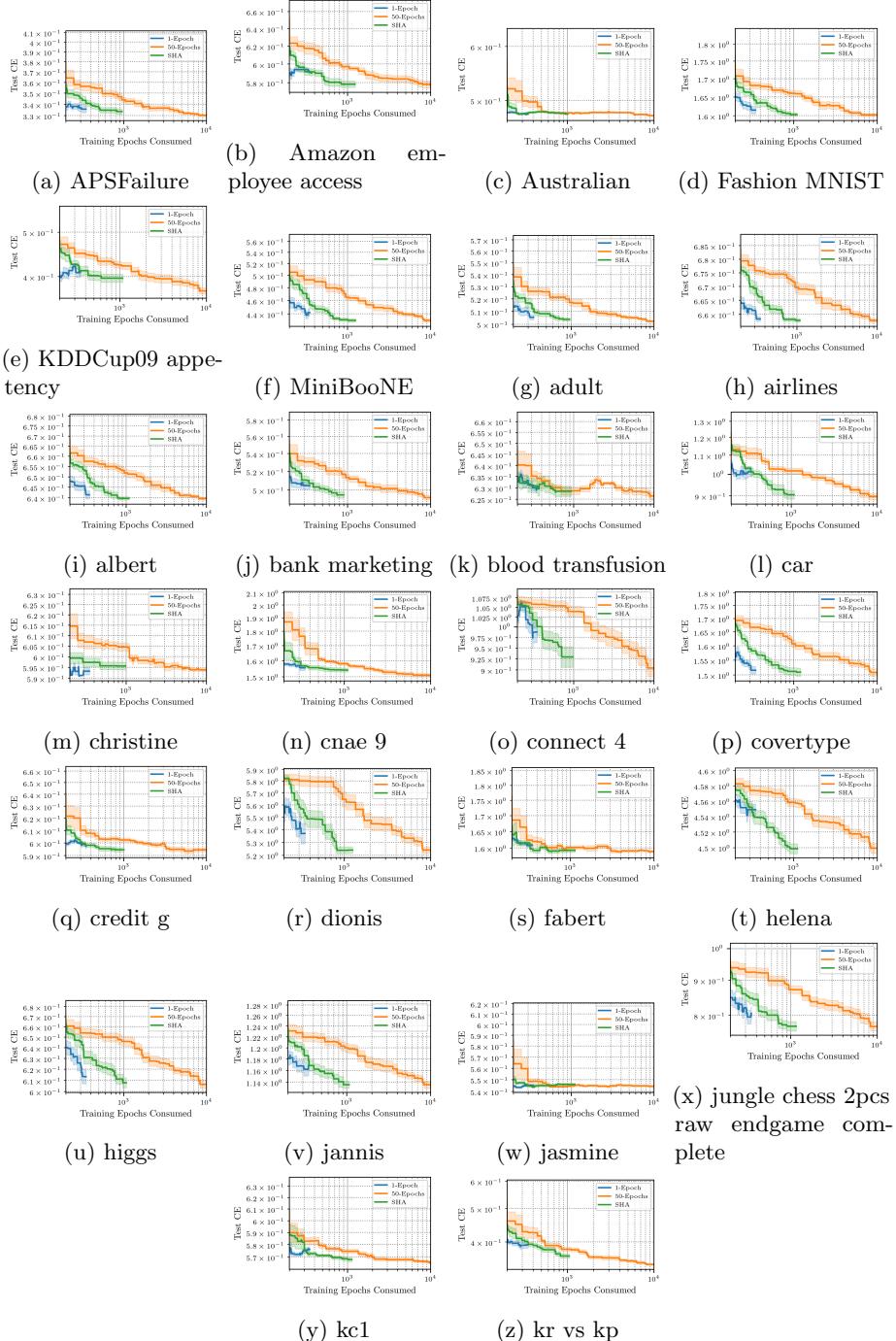


Fig. 19: Comparing pruners 1-Epoch, 50-Epochs and SHA on the LCBench benchmark (**First Part**).

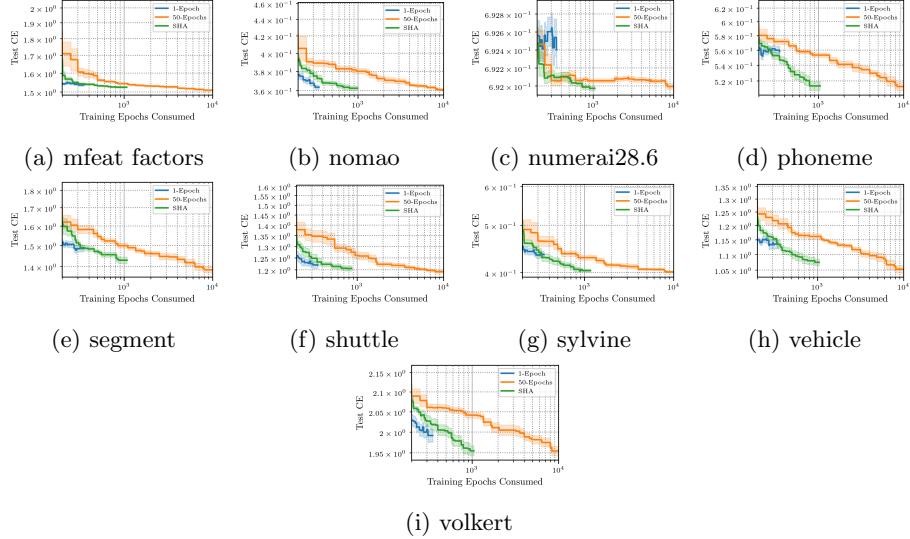


Fig. 20: Comparing pruners 1-Epoch, 50-Epochs and SHA on the LC-Bench benchmark (**Second Part**).

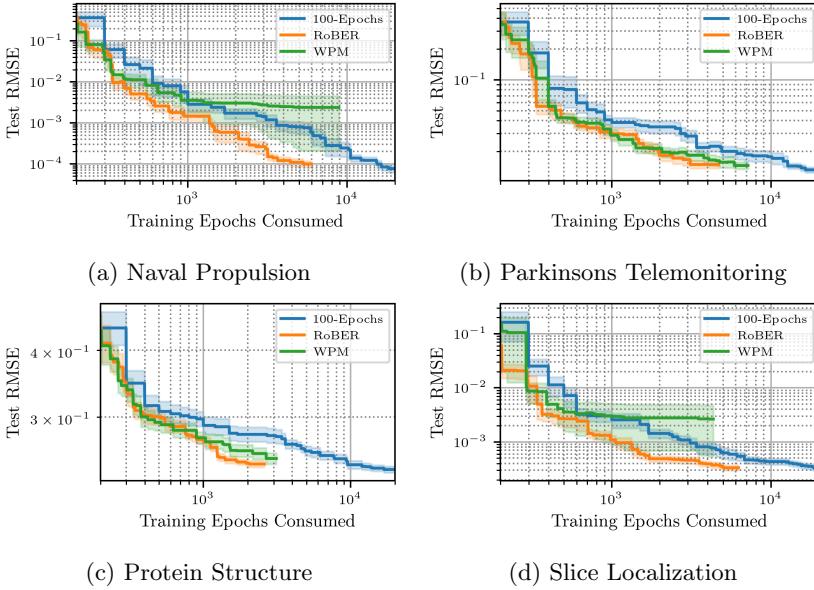


Fig. 21: Comparing RoBER with Weighted Probabilistic Model (WPM) on HPOBench. RoBER is consistently better on all the problems and the WPM is something very unstable and worse than the basic 100-Epochs evaluation strategy.

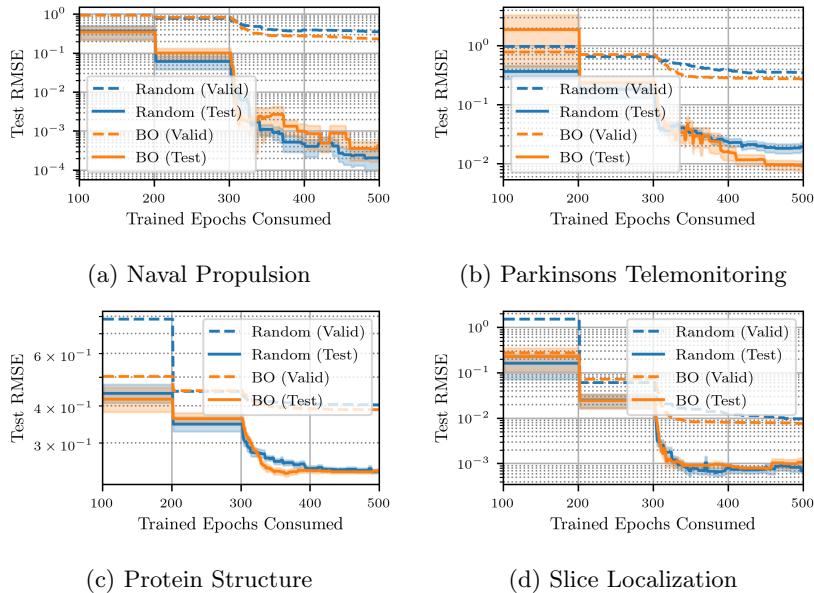


Fig. 22: Looking at the Generalization Error in HPO for the 1-Epoch multi-fidelity agent.