Supervising the Multi-Fidelity Race of Hyperparameter Configurations

Martin Wistuba[*] [†]
Amazon Web Services, Berlin, Germany
marwistu@amazon.com

Arlind Kadra[*]
University of Freiburg, Freiburg, Germany
kadraa@cs.uni-freiburg.de

Josif Grabocka
University of Freiburg, Freiburg, Germany
grabocka@cs.uni-freiburg.de

Abstract

Multi-fidelity (gray-box) hyperparameter optimization techniques (HPO) have recently emerged as a promising direction for tuning Deep Learning methods. However, existing methods suffer from a sub-optimal allocation of the HPO budget to the hyperparameter configurations. In this work, we introduce DyHPO, a Bayesian Optimization method that learns to decide which hyperparameter configuration to train further in a dynamic race among all feasible configurations. We propose a new deep kernel for Gaussian Processes that embeds the learning curve dynamics, and an acquisition function that incorporates multi-budget information. We demonstrate the significant superiority of DyHPO against state-of-the-art hyperparameter optimization methods through large-scale experiments comprising 50 datasets (Tabular, Image, NLP) and diverse architectures (MLP, CNN/NAS, RNN).

1 Introduction

Hyperparameter Optimization (HPO) is arguably an acute open challenge for Deep Learning (DL), especially considering the crucial impact HPO has on achieving state-of-the-art empirical results. Unfortunately, HPO for DL is a relatively under-explored field and most DL researchers still optimize their hyperparameters via obscure trial-and-error practices. On the other hand, traditional Bayesian Optimization HPO methods [Snoek et al., 2012, Bergstra et al., 2011] are not directly applicable to deep networks, due to the infeasibility of evaluating a large number of hyperparameter configurations. In order to scale HPO for DL, three main directions of research have been recently explored. *(i) Online HPO* methods search for hyperparameters during the optimization process via meta-level controllers [Chen et al., 2017, Parker-Holder et al., 2020], however, this online adaptation can not accommodate all hyperparameters (e.g. related to architectural changes). *(ii) Gradient-based HPO* techniques, on the other hand, compute the derivative of the validation loss w.r.t. hyperparameters by reversing the training update steps [Maclaurin et al., 2015, Franceschi et al., 2017, Lorraine et al., 2020], however, the reversion is not directly applicable to all cases (e.g. dropout rate). The last direction, *(iii) Gray-box HPO* techniques discard sub-optimal configurations after evaluating them on lower budgets [Li et al., 2017, Falkner et al., 2018].

In contrast to the online and gradient-based alternatives, gray-box approaches can be deployed in an off-the-shelf manner to all types of hyperparameters and architectures. The gray-box concept is based on the intuition that a poorly-performing hyperparameter configuration can be identified and

[*] equal contribution
[†] work does not relate to position at Amazon

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

terminated by inspecting the validation loss of the first few epochs, instead of waiting for the full convergence. The most prominent gray-box algorithm is Hyperband [Li et al., 2017], which is based on successive halving. It runs random configurations at different budgets (e.g. number of epochs) and successively halves these configurations by keeping only the top performers. Follow-up works, such as BOHB [Falkner et al., 2018] or DEHB [Awad et al., 2021], replace the random sampling of Hyperband with a sampling based on Bayesian optimization or differentiable evolution.

Despite their great practical potential, gray-box methods suffer from a major issue. The low-budget (few epochs) performances are not always a good indicator for the full-budget (full convergence) performances. For example, a properly regularized network converges slower in the first few epochs, however, typically performs better than a non-regularized variant after the full convergence. In other words, there can be a poor rank correlation of the configurations' performances at different budgets.

We introduce DyHPO, a Bayesian Optimization (BO) approach based on Gaussian Processes (GP), that proposes a novel treatment to the multi-budget (a.k.a. multi-fidelity) setup. In this perspective, we propose a deep kernel GP that captures the learning dynamics. As a result, we train a kernel capable of capturing the similarity of a pair of hyperparameter configurations, even if the pair's configurations are evaluated at different budgets. Furthermore, we extend Expected Improvement [Jones et al., 1998] to the multi-budget case, by introducing a new mechanism for the incumbent configuration of a budget.

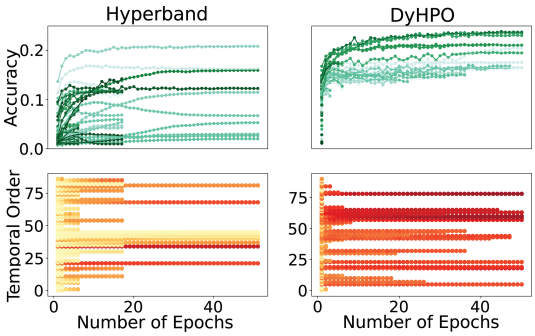

Figure 1: **Top:** The learning curve for different hyperparameter configurations. The darker the learning curve, the later it was evaluated during the search. **Bottom:** The hyperparameter indices in a temporal order as evaluated during the optimization and their corresponding curves.

We illustrate the differences between our racing strategy and successive halving with the experiment of Figure 1, where, we showcase the HPO progress of two different methods on the "Helena" dataset from the LCBench benchmark [Zimmer et al., 2021]. Hyperband [Li et al., 2017] is a gray-box approach that *statically* pre-allocates the budget for a set of candidates (Hyperband bracket) according to a predefined policy. However, DyHPO *dynamically* adapts the allocation of budgets for configurations after every HPO step (a.k.a. a dynamic race). As a result, DyHPO invests only a small budget on configurations that show little promise as indicated by the intermediate scores.

The joint effect of modeling a GP kernel across budgets together with a dedicated acquisition function leads to DyHPO achieving a statistically significant empirical gain against state-of-the-art gray-box baselines [Falkner et al., 2018, Awad et al., 2021], including prior work on multi-budget GPs [Kandasamy et al., 2017, 2020] or neural networks [Li et al., 2020b]. We demonstrate the performance of DyHPO in three diverse deep learning architectures (MLP, CNN/NAS, RNN) and 50 datasets of three diverse modalities (tabular, image, natural language processing). We believe our method is a step forward toward making HPO for DL practical and feasible. Overall, our contributions can be summarized as follows:

- We introduce a novel Bayesian surrogate for gray-box HPO optimization. Our novel surrogate model predicts the validation score of a machine learning model based on both the hyperparameter configuration, the budget information, and the learning curve.

- We derive a simple yet robust way to combine this surrogate model with Bayesian optimization, reusing most of the existing components currently used in traditional Bayesian optimization methods.

- Finally, we demonstrate the efficiency of our method for HPO and neural architecture search tasks compared to the current state-of-the-art methods in HPO, by outperforming seven strong HPO baselines with a statistically significant margin. As an overarching goal, we believe our method is an important step toward scaling HPO for DL.

2 Related Work on Gray-box HPO

Multi-Fidelity Bayesian Optimization and Bandits. Bayesian optimization is a black-box function optimization framework that has been successfully applied in optimizing hyperparameter and neural architectures alike [Snoek et al., 2012, Kandasamy et al., 2018, Bergstra et al., 2011]. To further improve Bayesian optimization, several works propose low-fidelity data approximations of hyperparameter configurations by training on a subset of the data [Swersky et al., 2013, Klein et al., 2017a], or by terminating training early [Swersky et al., 2014]. Additionally, several methods extend Bayesian optimization to multi-fidelity data by engineering new kernels suited for this problem [Swersky et al., 2013, 2014, Poloczek et al., 2017]. Kandasamy et al. [2016] extends GP-UCB [Srinivas et al., 2010] to the multi-fidelity setting by learning one Gaussian Process (GP) with a standard kernel for each fidelity. Their later work improves upon this method by learning one GP for all fidelities that enables the use of continuous fidelities [Kandasamy et al., 2017]. The work by Takeno et al. [2020] follows a similar idea but proposes to use an acquisition function based on information gain instead of UCB. While most of the works rely on GPs to model the surrogate function, Li et al. [2020b] use a Bayesian neural network that models the complex relationship between fidelities with stacked neural networks, one for each fidelity.

Hyperband [Li et al., 2017] is a bandits-based multi-fidelity method for hyperparameter optimization that selects hyperparameter configurations at random and uses successive halving [Jamieson and Talwalkar, 2016] with different settings to early-stop less promising training runs. Several improvements have been proposed to Hyperband with the aim to replace the random sampling of hyperparameter configurations with a more guided approach [Bertrand et al., 2017, Wang et al., 2018, Wistuba, 2017]. BOHB [Falkner et al., 2018] uses TPE [Bergstra et al., 2011] and builds a surrogate model for every fidelity adhering to a fixed-fidelity selection scheme. DEHB [Awad et al., 2021] samples candidates using differential evolution which handles large and discrete search spaces better than BOHB. Mendes et al. [2021] propose a variant of Hyperband which allows to skip stages.

Learning Curve Prediction A variety of methods attempt to extrapolate a partially observed learning curve in order to estimate the probability that a configuration will improve over the current best solution. Domhan et al. [2015] propose to ensemble a set of parametric functions to extrapolate a partial learning curve. While this method is able to extrapolate with a single example, it requires a relatively long learning curve to do so. The work by Klein et al. [2017b] build upon the idea of using a set of parametric functions. The main difference is that they use a heteroscedastic Bayesian model to learn the ensemble weights. Baker et al. [2018] propose to use support vector machines (SVM) as an auto-regressive model. The SVM predicts the next value of a learning curve, the original learning curve is augmented by this value and we keep predicting further values. The work by Gargiani et al. [2019] use a similar idea but makes prediction based on the last K observations only and uses probabilistic models. Wistuba and Pedapati [2020] propose to learn a prediction model across learning curves from different tasks to avoid the costly learning curve collection. In contrast to DYHPO, none of these methods selects configuration but is limited to deciding when to stop a running configuration.

Multi-Fidelity Acquisition Functions Klein et al. [2017a] propose an acquisition function which allows for selecting hyperparameter configurations and the dataset subset size. The idea is to reduce training time by considering only a smaller part of the training data. In contrast to EI_{MF}, this acquisition function is designed to select arbitrary subset sizes whereas EI_{MF} is intended to slowly increase the invested budget over time. Mendes et al. [2020] extend the work of Klein et al. [2017a] to take business constraints into account.

Deep Kernel Learning with Bayesian Optimization. We are among the first to use deep kernel learning with Bayesian optimization and to the best of our knowledge the first to use it for multi-fidelity Bayesian optimization. Rai et al. [2016] consider the use of a deep kernel instead of a manually designed kernel in the context of standard Bayesian optimization, but, limit their experimentation to synthetic data and do not consider its use for hyperparameter optimization. Perrone et al. [2018], Wistuba and Grabocka [2021] use a pre-trained deep kernel to warm start Bayesian optimization with meta-data from previous optimizations. The aforementioned approaches are multi-task or transfer learning methods that require the availability of meta-data from related tasks.

In contrast to prior work, we propose a method that introduces deep learning to multi-fidelity HPO with Bayesian Optimization, and captures the learning dynamics across fidelities/budgets, combined with an acquisition function that is tailored for the gray-box setup.

3 Dynamic Multi-Fidelity HPO

3.1 Preliminaries

Gray-Box Optimization. The gray-box HPO setting allows querying configurations with a smaller budget compared to the total maximal budget B. Thus, we can query from the response function $f : \mathcal{X} \times \mathbb{N} \rightarrow \mathbb{R}$ where $f_{i,j} = f(\mathbf{x}_i, j)$ is the response after spending a budget of j on configuration $\mathbf{x}_i$. As before, these observations are noisy and we observe $y_{i,j} = f(\mathbf{x}_i, j) + \varepsilon_j$ where $\varepsilon_j \sim \mathcal{N}(0, \sigma^2_{j,n})$. Please note, we assume that the budget required to query $f_{i,j+b}$ after having queried $f_{i,j}$ is only b. Furthermore, we use the learning curve $\mathbf{Y}_{i,j-1} = (y_{i,1}, \ldots, y_{i,j-1})$ when predicting $f_{i,j}$.

Gaussian Processes (GP). Given a training data set $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$, the Gaussian Process assumption is that y_i is a random variable and the joint distribution of all y_i is assumed to be multivariate Gaussian distributed as $\mathbf{y} \sim \mathcal{N}\left(m\left(\mathbf{X}\right), k\left(\mathbf{X}, \mathbf{X}\right)\right)$. Furthermore, $\mathbf{f}_*$ for test instances $\mathbf{x}_*$ are jointly Gaussian with $\mathbf{y}$ as:

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(m\left(\mathbf{X}, \mathbf{x}_*\right), \begin{pmatrix} \mathbf{K}_n & \mathbf{K}_* \\ \mathbf{K}_*^T & \mathbf{K}_{**} \end{pmatrix}\right) . \tag{1}$$

The mean function m is often set to $\mathbf{0}$ and its covariance function k depends on parameters $\boldsymbol{\theta}$. For notational convenience, we use $\mathbf{K}_n = k\left(\mathbf{X}, \mathbf{X}|\boldsymbol{\theta}\right) + \sigma_n^2 \mathbf{I}$, $\mathbf{K}_* = k\left(\mathbf{X}, \mathbf{X}_*|\boldsymbol{\theta}\right)$ and $\mathbf{K}_{**} = k\left(\mathbf{X}_*, \mathbf{X}_*|\boldsymbol{\theta}\right)$ to define the kernel matrices. We can derive the posterior predictive distribution with mean and covariance as follows:

$$\mathbb{E}\left[\mathbf{f}_*|\mathbf{X}, \mathbf{y}, \mathbf{X}_*\right] = \mathbf{K}_*^T \mathbf{K}_n^{-1} \mathbf{y}, \ \operatorname{cov}\left[\mathbf{f}_*|\mathbf{X}, \mathbf{X}_*\right] = \mathbf{K}_{**} - \mathbf{K}_*^T \mathbf{K}_n^{-1} \mathbf{K}_* \tag{2}$$

Often, the kernel function is manually engineered, one popular example is the squared exponential kernel. However, in this work, we make use of the idea of deep kernel learning [Wilson et al., 2016]. The idea is to model the kernel as a neural network φ and learn the best kernel transformation $\mathbf{K}\left(\theta, w\right) := k(\varphi(\mathbf{x}, w), \varphi(\mathbf{x}'; w)|\boldsymbol{\theta})$, which allows us to use convolutional operations in our kernel.

3.2 Deep Multi-Fidelity Surrogate

We propose to use a Gaussian Process surrogate model that infers the value of $f_{i,j}$ based on the hyperparameter configuration $\mathbf{x}_i$, the budget j as well as the past learning curve $\mathbf{Y}_{i,j-1}$. For this purpose, we use a deep kernel as:

$$\mathbf{K}\left(\theta, w\right) := k(\varphi(\mathbf{x}_i, \mathbf{Y}_{i,j-1}, j; w), \varphi(\mathbf{x}_{i'}, \mathbf{Y}_{i',j'-1}, j'; w); \theta) \tag{3}$$

We use a squared exponential kernel for k and the neural network φ is composed of linear and convolutional layers as shown in Figure 2. We normalize the budget j to a range between 0 and 1 by dividing it by the maximum budget B. Afterward, it is concatenated with the hyperparameter configuration $\mathbf{x}_i$ and fed to a linear layer. The learning curve $\mathbf{Y}_{i,j-1}$ is transformed by a one-dimensional convolution followed by a global max pooling layer. Finally, both representations are fed to another linear layer.

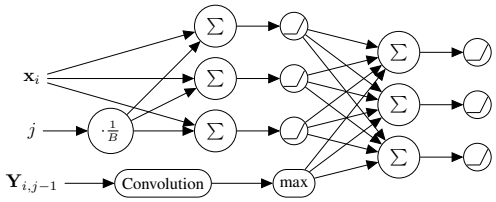

Figure 2: The feature extractor φ of our kernel.

Its output will be the input to the kernel function k. Both, the kernel k and the neural network φ consist of trainable parameters $\boldsymbol{\theta}$ and $\mathbf{w}$, respectively. We find their optimal values by computing the maximum likelihood estimates as:

$$\hat{\theta}, \hat{w} \ = \ \arg\max_{\theta, w} p(\mathbf{y}|\mathbf{X}, \mathbf{Y}, \theta, w) \propto \arg\min_{\theta, w} \mathbf{y}^{\mathrm{T}} \mathbf{K}\left(\theta, w\right)^{-1} \mathbf{y} + \log\left|\mathbf{K}\left(\theta, w\right)\right| \tag{4}$$

In order to solve this optimization problem, we use gradient descent and Adam [Kingma and Ba, 2015] with a learning rate of 0.1. Given the maximum likelihood estimates, we can approximate the predictive posterior through $p\left(f_{i,j}|\mathbf{x}_i, \mathbf{Y}_{i,j-1}, j, \mathcal{D}, \hat{\boldsymbol{\theta}}, \hat{\mathbf{w}}\right)$, and ultimately compute the mean and covariance of this Gaussian using Equation 2.

3.3 Multi-Fidelity Expected Improvement

Expected improvement [Jones et al., 1998] is a commonly used acquisition function and is defined as:

$$\mathrm{EI}(\mathbf{x}|\mathcal{D}) = \mathbb{E}\left[\max\left\{f(\mathbf{x}) - y^{\max}, 0\right\}\right] \ , \tag{5}$$

where $y^{\max}$ is the largest observed value of f. We propose a multi-fidelity version of it as:

$$\mathrm{EI}_{\mathrm{MF}}(\mathbf{x}, j|\mathcal{D}) = \mathbb{E}\left[\max\left\{f(\mathbf{x}, j) - y_j^{\max}, 0\right\}\right] \ , \tag{6}$$

where:

$$y_j^{\max} = \begin{cases} \max\left\{y \mid ((\mathbf{x}, \cdot, j), y) \in \mathcal{D}\right\} & \text{if } ((\mathbf{x}, \cdot, j), y) \in \mathcal{D} \\ \max\left\{y \mid (\cdot, y) \in \mathcal{D}\right\} & \text{otherwise} \end{cases} \tag{7}$$

Simply put, $y_j^{\max}$ is the largest observed value of f for a budget of j if it exists already, otherwise, it is the largest observed value for any budget. If there is only one possible budget, the multi-fidelity expected improvement is identical to expected improvement.

3.4 The DyHPO Algorithm

The DyHPO algorithm looks very similar to many black-box Bayesian optimization algorithms as shown in Algorithm 1. The big difference is that at each step we dynamically decide which candidate configuration to train *for a small additional budget*.

Algorithm 1 DyHPO Algorithm

1: $b(\mathbf{x}) = 0 \ \forall \mathbf{x} \in \mathcal{X}$
2: **while** not converged **do**
3: $\quad \mathbf{x}_i \leftarrow \arg\max_{\mathbf{x} \in \mathcal{X}} \mathrm{EI}_{\mathrm{MF}}\left(\mathbf{x}, b(\mathbf{x}) + 1\right)$ (Sec. 3.3)
4: $\quad$ Observe $y_{i,b(\mathbf{x}_i)+1}$.
5: $\quad b(\mathbf{x}_i) \leftarrow b(\mathbf{x}_i) + 1$
6: $\quad \mathcal{D} \leftarrow \mathcal{D} \cup \left\{\left((\mathbf{x}_i, \mathbf{Y}_{i,b(\mathbf{x}_i)-1}, b(\mathbf{x}_i)), y_{i,b(\mathbf{x}_i)}\right)\right\}$
7: $\quad$ Update the surrogate on $\mathcal{D}$. (Sec. 3.2)
return $\mathbf{x}_i$ with largest $y_{i,\cdot}$.

Possible candidates are previously unconsidered configurations as well as configurations that did not reach the maximum budget. In Line 2, the most promising candidate is chosen using the acquisition function introduced in Section 3.3 and the surrogate model's predictions. It is important to highlight that we do not maximize the acquisition function along the budget dimensionality. Instead, we set the budget b such that it is by exactly one higher than the budget used to evaluate $\mathbf{x}_i$ before. This ensures that we explore configurations by slowly increasing the budget. After the candidate and the corresponding budget are selected, the function f is evaluated and we observe $y_{i,j}$ (Line 3). This additional data point is added to $\mathcal{D}$ in Line 4. Then in Line 5, the surrogate model is updated according to the training scheme described in Section 3.2.

4 Experimental Protocol

4.1 Experimental Setup

We evaluate DyHPO in three different settings on hyperparameter optimization for tabular, text, and image classification against several competitor methods, the details of which are provided in the following subsections. We ran all of our experiments on an Amazon EC2 M5 Instance (m5.xlarge). In our experiments, we report the mean of ten repetitions and we report two common metrics, the regret and the average rank. The regret refers to the absolute difference between the score of the solution found by an optimizer compared to the best possible score. If we report the regret as an aggregate result over multiple datasets, we report the mean over all regrets. The average rank is the

metric we use to aggregate rank results over different datasets. We provide further implementation and training details in Appendix A.4. Our implementation of DyHPO is publicly available.[3]

4.2 Benchmarks

In our experiments, we use the following benchmarks. We provide more details in Appendix A.1.

LCBench: A learning curve benchmark [Zimmer et al., 2021] that evaluates neural network architectures for tabular datasets. LCBench contains learning curves for 35 different datasets, where 2,000 neural networks per dataset are trained for 50 epochs with Auto-PyTorch.

TaskSet: A benchmark that features diverse tasks Metz et al. [2020] from different domains and includes 5 search spaces with different degrees of freedom, where, every search space includes 1000 hyperparameter configurations. In this work, we focus on a subset of NLP tasks (12 tasks) and we use the Adam8p search space with 8 continuous hyperparameters.

NAS-Bench-201: A benchmark consisting of 15625 hyperparameter configurations representing different architectures on the CIFAR-10, CIFAR-100 and ImageNet datasets Dong and Yang [2020]. NAS-Bench-201 features a search space of 6 categorical hyperparameters and each architecture is trained for 200 epochs.

4.3 Baselines

Random Search: A random/stochastic black-box search method for HPO.

HyperBand: A multi-arm bandit method that extends successive halving by multiple brackets with different combinations of the initial number of configurations, and their initial budget [Li et al., 2017].

BOHB: An extension of Hyperband that replaces the random sampling of the initial configurations for each bracket with recommended configurations from a model-based approach [Falkner et al., 2018]. BOHB builds a model for every fidelity that is considered.

DEHB: A method that builds upon Hyperband by exploiting differential evolution to sample the initial candidates of a Hyperband bracket [Awad et al., 2021].

ASHA: An asynchronous version of successive halving (or an asynchronous version of Hyperband if multiple brackets are run). ASHA Li et al. [2020a] does not wait for all configurations to finish inside a successive halving bracket, but, instead promotes configurations to the next successive halving bracket in real-time.

MF-DNN: A multi-fidelity Bayesian optimization method that uses deep neural networks to capture the relationships between different fidelities Li et al. [2020b].

Dragonfly: We compare against BOCA [Kandasamy et al., 2017] by using the Dragonfly library Kandasamy et al. [2020]. This method suggests the next hyperparameter configuration as well as the budget it should be evaluated for.

4.4 Research Hypotheses and Associated Experiments

Hypothesis 1: DyHPO achieves state-of-the-art results in multi-fidelity HPO.

Experiment 1: We compare DyHPO against the baselines of Section 4.3 on the benchmarks of Section 4.2 with the experimental setup of Section 4.1. For TaskSet we follow the authors' recommendation and report the number of steps (every 200 iterations).

Hypothesis 2: DyHPO's runtime overhead has a negligible impact on the quality of results.

Experiment 2: We compare DyHPO against the baselines of Section 4.3 over the wallclock time. The wallclock time includes both *(i)* the optimizer's runtime overhead for recommending the next hyperparameter configuration, plus *(ii)* the time needed to evaluate the recommended configuration. In this experiment, we consider all datasets where the average training time per epoch is at least 10 seconds, because, for tasks where the training time is short, there is no practical justification for

[3] `https://github.com/releaunifreiburg/DyHPO`

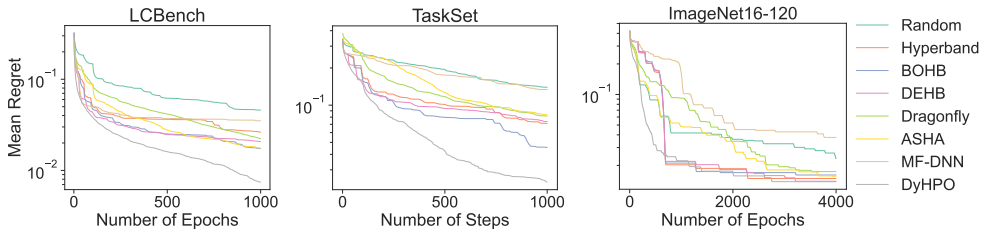

Figure 3: The mean regret for the different benchmarks over the number of epochs or steps (every 200 iterations). The results are aggregated over 35 different datasets for LCBench and aggregated over 12 different NLP tasks for TaskSet.

complex solutions and their overhead. In these cases, we recommend using a random search. We don't report results for TaskSet because the benchmark lacks training times.

Hypothesis 3: DyHPO uses the computational budget more efficiently than baselines.

Experiment 3: To further verify that DyHPO is efficient compared to the baselines, we investigate whether competing methods spend their budgets on qualitative candidates. Concretely we: i) calculate the precision of the top (w.r.t. ground truth) performing configurations that were selected by each method across different budgets, ii) compute the average regret of the selected configurations across budget, and iii) we compare the fraction of top-performing configurations at a given budget that were not top performers at lower budgets, i.e. measure the ability to handle the poor correlation of performances across budgets.

5 Results

Experiment 1: DyHPO achieves state-of-the-art results. In our first experiment, we evaluate the various methods on the benchmarks listed in Section 4.2. We show the aggregated results in Figure 3, the results show that DyHPO manages to outperform competitor methods over the set of considered benchmarks by achieving a better mean regret across datasets. Not only does DyHPO achieve a better final performance, it also achieves strong anytime results by converging faster than the competitor methods. For the extended results, related to the performance of all methods on a dataset level, we refer the reader to Appendix B.

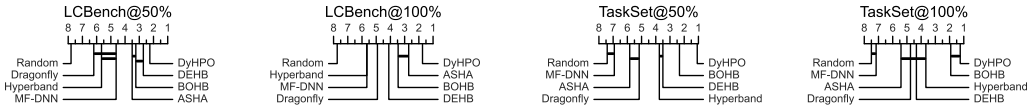

Figure 4: Critical difference diagram for LCBench and TaskSet in terms of the number of HPO steps. The results correspond to results after 500 and 1,000 epochs. Connected ranks via a bold bar indicate that performances are not significantly different ($p > 0.05$).

In Figure 4, we provide further evidence that DyHPO's improvement over the baselines is statistically significant. The critical difference diagram presents the ranks of all methods and provides information on the pairwise statistical difference between all methods for two fractions of the number of HPO steps (50% and 100%). We included the LCBench and TaskSet benchmarks in our significance plots. NAS-Bench-201 was omitted because it has only 3 datasets and the statistical test cannot be applied. Horizontal lines indicate groupings of methods that are not significantly different. As suggested by the best published practices Demsar [2006], we use the Friedman test to reject the null hypothesis followed by a pairwise post-hoc analysis based on the Wilcoxon signed-rank test ($\alpha = 0.05$).

For LCBench, DyHPO already outperforms the baselines significantly after 50% of the search budget, with a statistically significant margin. As the optimization procedure continues, DyHPO manages to extend its gain in performance and is the only method that has a statistically significant improvement against all the other competitor methods. Similarly, for TaskSet, DyHPO manages to outperform all methods with a statistically significant margin only halfway through the optimization procedure and achieves the best rank over all methods. However, as the optimization procedure continues, BOHB

manages to decrease the performance gap with DyHPO, although, it still achieves a worse rank across all datasets. Considering the empirical results, we conclude that **Hypothesis 1 is validated and that DyHPO achieves state-of-the-art results on multi-fidelity HPO**.

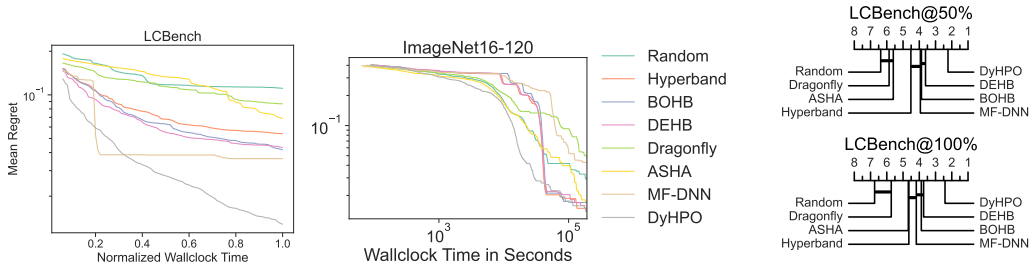

Figure 5: **Left:** The regret over time for all methods during the optimization procedure for the LCBench benchmark and the ImageNet dataset from the NAS-Bench-201 benchmark. The normalized wall clock time represents the actual run time divided by the total wall clock time of DyHPO including the overhead of fitting the deep GP. **Right:** The critical difference diagram for LCBench halfway through the HPO wall-clock time, and in the end. Connected ranks via a bold bar indicate that differences are not significant ($p > 0.05$).

Experiment 2: On the impact of DyHPO's overhead on the results. We present the results of our second experiment in Figure 5 (left), where, as it can be seen, DyHPO still outperforms the other methods when its overhead is considered. For LCBench, DyHPO manages to get an advantage fairly quickly and it only increases the gap in performance with the other methods as the optimization process progresses. Similarly, in the case of ImageNet from NAS-Bench-201, DyHPO manages to gain an advantage earlier than other methods during the optimization procedure. Although in the end DyHPO still performs better than all the other methods, we believe most of the methods converge to a good solution and the differences in the final performance are negligible. For the extended results, related to the performance of all methods on a dataset level over time, we refer the reader to the plots in Appendix B. Additionally, in Figure 5 (right), we provide the critical difference diagrams for LCBench that present the ranks and the statistical difference of all methods halfway through the optimization procedure, and in the end. As it can be seen, DyHPO has a better rank with a significant margin with only half of the budget used and it retains the advantage until the end.

Experiment 3: On the efficiency of DyHPO. In Figure 6 (left), we plot the precision of every method for different budgets during the optimization procedure, which demonstrates that DyHPO effectively explores the search space and identifies promising candidates. The precision at an epoch i is defined as the number of top 1% candidates that are trained, divided by the number of all candidates trained, both trained for at least i epochs. The higher the precision, the more relevant candidates were considered and the less computational resources were wasted. For small budgets, the precision is low since DyHPO spends budget to consider various candidates, but then, promising candidates are successfully identified and the precision quickly increases. This argument is further supported in Figure 6 (middle), where we visualize the average regret of all the candidates trained for at least the specified number of epochs on the x-axis. In contrast to the regret plots, here we do not show the regret of the best configuration, but the mean regret of all the selected configurations. The analysis deduces a similar finding, our method DyHPO selects more qualitative hyperparameter configurations than all the baselines.

An interesting property of multi-fidelity HPO is the phenomenon of poor rank correlations among the validation performance of candidates at different budgets. In other words, a configuration that achieves a poor performance at a small budget can perform better at a larger budget. To analyze this phenomenon, we measure the percentage of "good" configurations at a particular budget, that were "bad" performers in at least one of the smaller budgets. We define a "good" performance at a budget B when a configuration achieves a validation accuracy ranked among the top 1/3 of the validation accuracies belonging to all the other configurations that were run until that budget B.

In Figure 6 (right), we analyze the percentage of "good" configurations at each budget denoted by the x-axis, that were "bad" performers in at least one of the lower budgets. Such a metric is a proxy for the degree of the promotion of "bad" configurations towards higher budgets. We present the analysis for all the competing methods of our experimental protocol from Section 4. We have additionally included the ground-truth line annotated as "Baseline", which represents the fraction of past poor

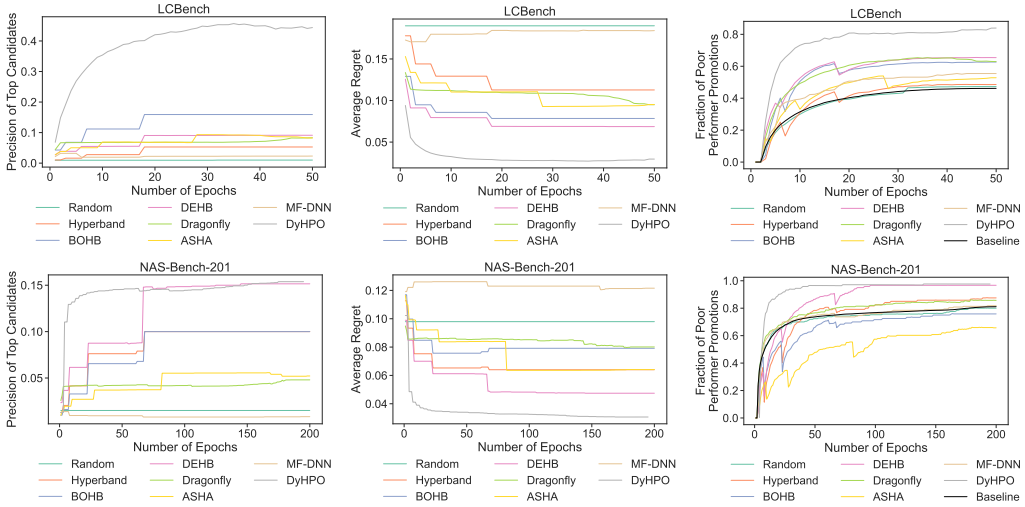

Figure 6: The efficiency of DyHPO as the optimization progresses. **Left:** The fraction of top-performing candidates from all candidates that were selected to be trained. **Middle:** The average regret for the configurations that were selected to be trained at a given budget. **Right:** The percentage of configurations that belong to the top 1/3 configurations at a given budget and that were in the top bottom 2/3 of the configurations at a previous budget. All of the results are from the LCBench and NAS-Bench-201 benchmark.

performers among all the feasible configurations in the search space. In contrast, the respective methods compute the fraction of promotions only among the configurations that those methods have considered (i.e. selected within their HPO trials) until the budget indicated by the x-axis. We see that there is a high degree of "good" configurations that were "bad" at a previous budget, with fractions of the ground-truth "Baseline" going up to 40% for the LCBench benchmark and up to 80% for the NAS-Bench-201 benchmark.

On the other hand, the analysis demonstrates that our method DyHPO has promoted more "good" configurations that were "bad" in a lower budget, compared to all the rival methods. In particular, more than 80% of selected configurations from the datasets belonging to either benchmark were "bad" performers at a lower budget. The empirical evidence validates **Hypothesis 3 and demonstrates that DyHPO efficiently explores qualitative candidates.** We provide the results of our analysis for DyHPO's efficiency on the additional benchmarks (Taskset) in Appendix B.

Ablating the impact of the learning curve

One of the main differences between DyHPO and similar methods Kandasamy et al. [2017], is that the learning curve is an input to the kernel function. For this reason, we investigate the impact of this design choice. We consider a variation of DyHPO w/o CNN, which is simply DyHPO without the learning curve.

It is worth emphasizing that both variants (with and without the learning curve) are multi-fidelity surrogates and both receive the budget information through the inputted index j in Equation 3. The only difference is that DyHPO additionally incorporates the pattern of the learning curve.

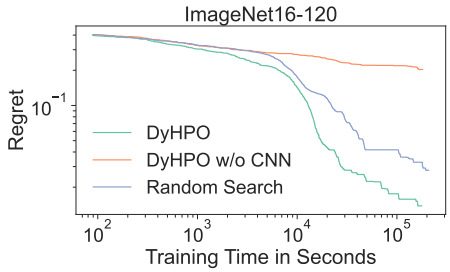

Figure 7: Ablating the impact of the learning curve on DyHPO.

We run the ablation on the NAS-Bench-201 benchmark and report the results for ImageNet, the largest dataset in our collection. The ablation results are shown in Figure 7, while the remaining results on the other datasets are shown in Figure 8 of the appendix. Based on the results from our

learning curve ablation, we conclude that the use of an explicit learning curve representation leads to significantly better results.

6 Limitations of Our Method

Although DyHPO shows a convincing and statistically significant reduction of the HPO time on diverse Deep Learning (DL) experiments, we cautiously characterized our method only as a "step towards" scaling HPO for DL. The reason for our restrain is the lack of tabular benchmarks for HPO on very large deep learning models, such as Transformers-based architectures [Devlin et al., 2019]. Additionally, the pause and resume part of our training procedure can only be applied when tuning the hyperparameters of parametric models, otherwise, the training of a hyperparameter configuration would have to be restarted. Lastly, for small datasets that can be trained fast, the overhead of model-based techniques would make an approach like random search more appealing.

7 Conclusions

In this work, we present DyHPO, a new Bayesian optimization (BO) algorithm for the gray-box setting. We introduced a new surrogate model for BO that uses a learnable deep kernel and takes the learning curve as an explicit input. Furthermore, we motivated a variation of expected improvement for the multi-fidelity setting. Finally, we compared our approach on diverse benchmarks on a total of 50 different tasks against the current state-of-the-art methods on gray-box hyperparameter optimization (HPO). Our method shows significant gains and has the potential to become the de facto standard for HPO in Deep Learning.

Acknowledgments

Josif Grabocka and Arlind Kadra would like to acknowledge the grant awarded by Eva-Mayr-Stihl Stiftung. In addition, this research was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under grant number 417962828. Furthermore, the authors acknowledge support by the state of Baden-Württemberg through bwHPC and the German Research Foundation (DFG) through grant no INST 39/963-1 FUGG (bwForCluster NEMO).

References

Noor H. Awad, Neeratyoy Mallik, and Frank Hutter. DEHB: evolutionary hyberband for scalable, robust and efficient hyperparameter optimization. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, pages 2147–2153, 2021. doi: 10.24963/ijcai.2021/296. URL `https://doi.org/10.24963/ijcai.2021/296`.

Bowen Baker, Otkrist Gupta, Ramesh Raskar, and Nikhil Naik. Accelerating neural architecture search using performance prediction. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings*, 2018. URL `https://openreview.net/forum?id=HJqk3N1vG`.

James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain*, pages 2546–2554, 2011. URL `https://proceedings.neurips.cc/paper/2011/hash/86e8f7ab32cfd12577bc2619bc635690-Abstract.html`.

Hadrien Bertrand, Roberto Ardon, Matthieu Perrot, and Isabelle Bloch. Hyperparameter optimization of deep neural networks: Combining hyperband with bayesian model selection. In *Conférence sur l'Apprentissage Automatique*, 2017.

Yutian Chen, Matthew W. Hoffman, Sergio Gomez Colmenarejo, Misha Denil, Timothy P. Lillicrap, Matthew Botvinick, and Nando de Freitas. Learning to learn without gradient descent by gradient descent. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017,*

Sydney, NSW, Australia, 6-11 August 2017, pages 748–756, 2017. URL http://proceedings.mlr.press/v70/chen17e.html.

Janez Demsar. Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.*, 7: 1–30, 2006. URL http://jmlr.org/papers/v7/demsar06a.html.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186, 2019. doi: 10.18653/v1/n19-1423. URL https://doi.org/10.18653/v1/n19-1423.

Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In Qiang Yang and Michael J. Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 3460–3468. AAAI Press, 2015. URL http://ijcai.org/Abstract/15/487.

Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, 2020. URL https://openreview.net/forum?id=HJxyZkBKDr.

Stefan Falkner, Aaron Klein, and Frank Hutter. BOHB: robust and efficient hyperparameter optimization at scale. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 1436–1445, 2018. URL http://proceedings.mlr.press/v80/falkner18a.html.

Luca Franceschi, Michele Donini, Paolo Frasconi, and Massimiliano Pontil. Forward and reverse gradient-based hyperparameter optimization. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 1165–1173, 2017. URL http://proceedings.mlr.press/v70/franceschi17a.html.

Jacob R. Gardner, Geoff Pleiss, Kilian Q. Weinberger, David Bindel, and Andrew Gordon Wilson. Gpytorch: Blackbox matrix-matrix gaussian process inference with GPU acceleration. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 7587–7597, 2018. URL https://proceedings.neurips.cc/paper/2018/hash/27e8e17134dd7083b050476733207ea1-Abstract.html.

Matilde Gargiani, Aaron Klein, Stefan Falkner, and Frank Hutter. Probabilistic rollouts for learning curve extrapolation across hyperparameter settings. *CoRR*, abs/1910.04522, 2019. URL http://arxiv.org/abs/1910.04522.

Kevin G. Jamieson and Ameet Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, AISTATS 2016, Cadiz, Spain, May 9-11, 2016*, pages 240–248, 2016. URL http://proceedings.mlr.press/v51/jamieson16.html.

Donald R. Jones, Matthias Schonlau, and William J. Welch. Efficient global optimization of expensive black-box functions. *J. Global Optimization*, 13(4):455–492, 1998. doi: 10.1023/A:1008306431147. URL https://doi.org/10.1023/A:1008306431147.

Kirthevasan Kandasamy, Gautam Dasarathy, Junier B. Oliva, Jeff G. Schneider, and Barnabás Póczos. Gaussian process bandit optimisation with multi-fidelity evaluations. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 992–1000, 2016. URL https://proceedings.neurips.cc/paper/2016/hash/605ff764c617d3cd28dbbdd72be8f9a2-Abstract.html.

Kirthevasan Kandasamy, Gautam Dasarathy, Jeff G. Schneider, and Barnabás Póczos. Multi-fidelity bayesian optimisation with continuous approximations. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 1799–1808, 2017. URL http://proceedings.mlr.press/v70/kandasamy17a.html.

Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabás Póczos, and Eric P. Xing. Neural architecture search with bayesian optimisation and optimal transport. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 2020–2029, 2018. URL https://proceedings.neurips.cc/paper/2018/hash/f33ba15effa5c10e873bf3842afb46a6-Abstract.html.

Kirthevasan Kandasamy, Karun Raju Vysyaraju, Willie Neiswanger, Biswajit Paria, Christopher R. Collins, Jeff Schneider, Barnabás Póczos, and Eric P. Xing. Tuning hyperparameters without grad students: Scalable and robust bayesian optimisation with dragonfly. *J. Mach. Learn. Res.*, 21: 81:1–81:27, 2020. URL http://jmlr.org/papers/v21/18-223.html.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

Aaron Klein, Stefan Falkner, Simon Bartels, Philipp Hennig, and Frank Hutter. Fast bayesian optimization of machine learning hyperparameters on large datasets. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017, 20-22 April 2017, Fort Lauderdale, FL, USA*, pages 528–536, 2017a. URL http://proceedings.mlr.press/v54/klein17a.html.

Aaron Klein, Stefan Falkner, Jost Tobias Springenberg, and Frank Hutter. Learning curve prediction with bayesian neural networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017b. URL https://openreview.net/forum?id=S11KBYclx.

Liam Li, Kevin G. Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Jonathan Ben-tzur, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. A system for massively parallel hyperparameter tuning. In Inderjit S. Dhillon, Dimitris S. Papailiopoulos, and Vivienne Sze, editors, *Proceedings of Machine Learning and Systems 2020, MLSys 2020, Austin, TX, USA, March 2-4, 2020*. mlsys.org, 2020a. URL https://proceedings.mlsys.org/book/303.pdf.

Lisha Li, Kevin G. Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *J. Mach. Learn. Res.*, 18: 185:1–185:52, 2017. URL http://jmlr.org/papers/v18/16-558.html.

Shibo Li, Wei Xing, Robert M. Kirby, and Shandian Zhe. Multi-fidelity bayesian optimization via deep neural networks. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020b. URL https://proceedings.neurips.cc/paper/2020/hash/60e1deb043af37db5ea4ce9ae8d2c9ea-Abstract.html.

Jonathan Lorraine, Paul Vicol, and David Duvenaud. Optimizing millions of hyperparameters by implicit differentiation. In *The 23rd International Conference on Artificial Intelligence and Statistics, AISTATS 2020, 26-28 August 2020, Online [Palermo, Sicily, Italy]*, pages 1540–1552, 2020. URL http://proceedings.mlr.press/v108/lorraine20a.html.

Dougal Maclaurin, David Duvenaud, and Ryan P. Adams. Gradient-based hyperparameter optimization through reversible learning. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 2113–2122, 2015. URL http://proceedings.mlr.press/v37/maclaurin15.html.

Pedro Mendes, Maria Casimiro, Paolo Romano, and David Garlan. Trimtuner: Efficient optimization of machine learning jobs in the cloud via sub-sampling. In *28th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, MASCOTS 2020, Nice, France, November 17-19, 2020*, pages 1–8. IEEE, 2020. doi: 10.1109/MASCOTS50786.2020.9285971. URL https://doi.org/10.1109/MASCOTS50786.2020.9285971.

Pedro Mendes, Maria Casimiro, and Paolo Romano. Hyperjump: Accelerating hyperband via risk modelling. *CoRR*, abs/2108.02479, 2021. URL https://arxiv.org/abs/2108.02479.

Luke Metz, Niru Maheswaranathan, Ruoxi Sun, C. Daniel Freeman, Ben Poole, and Jascha Sohl-Dickstein. Using a thousand optimization tasks to learn hyperparameter search strategies. *CoRR*, abs/2002.11887, 2020. URL https://arxiv.org/abs/2002.11887.

Jack Parker-Holder, Vu Nguyen, and Stephen J. Roberts. Provably efficient online hyperparameter optimization with population-based bandits. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/c7af0926b294e47e52e46cfebe173f20-Abstract.html.

Valerio Perrone, Rodolphe Jenatton, Matthias W. Seeger, and Cédric Archambeau. Scalable hyperparameter transfer learning. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 6846–6856, 2018. URL https://proceedings.neurips.cc/paper/2018/hash/14c879f3f5d8ed93a09f6090d77c2cc3-Abstract.html.

Matthias Poloczek, Jialei Wang, and Peter I. Frazier. Multi-information source optimization. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 4288–4298, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/df1f1d20ee86704251795841e6a9405a-Abstract.html.

Akshara Rai, Ruta Desai, and Siddharth Goyal. Bayesian optimization with a neural network kernel, 2016. URL http://www.cs.cmu.edu/~rutad/files/BO_NN.pdf.

Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, pages 2960–2968, 2012. URL https://proceedings.neurips.cc/paper/2012/hash/05311655a15b75fab86956663e1819cd-Abstract.html.

Niranjan Srinivas, Andreas Krause, Sham M. Kakade, and Matthias W. Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, pages 1015–1022, 2010. URL https://icml.cc/Conferences/2010/papers/422.pdf.

Kevin Swersky, Jasper Snoek, and Ryan Prescott Adams. Multi-task bayesian optimization. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, pages 2004–2012, 2013. URL https://proceedings.neurips.cc/paper/2013/hash/f33ba15effa5c10e873bf3842afb46a6-Abstract.html.

Kevin Swersky, Jasper Snoek, and Ryan Prescott Adams. Freeze-thaw bayesian optimization. *CoRR*, abs/1406.3896, 2014. URL http://arxiv.org/abs/1406.3896.

Shion Takeno, Hitoshi Fukuoka, Yuhki Tsukada, Toshiyuki Koyama, Motoki Shiga, Ichiro Takeuchi, and Masayuki Karasuyama. Multi-fidelity bayesian optimization with max-value entropy search and its parallelization. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, pages 9334–9345, 2020. URL http://proceedings.mlr.press/v119/takeno20a.html.

Jiazhuo Wang, Jason Xu, and Xuejun Wang. Combination of hyperband and bayesian optimization for hyperparameter optimization in deep learning. *CoRR*, abs/1801.01596, 2018. URL http://arxiv.org/abs/1801.01596.

Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P. Xing. Deep kernel learning. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, AISTATS 2016, Cadiz, Spain, May 9-11, 2016*, pages 370–378, 2016. URL http://proceedings.mlr.press/v51/wilson16.html.

Martin Wistuba. Bayesian optimization combined with incremental evaluation for neural network architecture optimization. In *AutoML@PKDD/ECML*, 2017.

Martin Wistuba and Josif Grabocka. Few-shot bayesian optimization with deep kernel surrogates. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*, 2021. URL `https://openreview.net/forum?id=bJxgv5C3sYc`.

Martin Wistuba and Tejaswini Pedapati. Learning to rank learning curves. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 10303–10312. PMLR, 2020. URL `http://proceedings.mlr.press/v119/wistuba20a.html`.

Lucas Zimmer, Marius Lindauer, and Frank Hutter. Auto-pytorch: Multi-fidelity metalearning for efficient and robust autodl. *IEEE Trans. Pattern Anal. Mach. Intell.*, 43(9):3079–3090, 2021. doi: 10.1109/TPAMI.2021.3067763. URL `https://doi.org/10.1109/TPAMI.2021.3067763`.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 6.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See Section "Societal Implications" in the Appendix.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No] We provide our main algorithm in Section 3 and we additionally provide the detailed implementation details in Appendix A for all methods and benchmarks. We will release the code for the camera-ready version of our work.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] Please see Appendix A.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] We report the statistical significance of the performance difference between our method and the baselines in Section 5
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Section 4.1.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] See Section 4.2 and Section 4.3.
   (b) Did you mention the license of the assets? [Yes] See Appendix A.1 and A.5 where we provide references to the assets where the license is included.
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] The benchmarks and baselines are open-sourced.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] The data does not contain personally identifiable information or offensive content.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Societal Implications

In our work, we use only publicly available data with no privacy concerns. Furthermore, our algorithm reduces the overall time for fitting deep networks, therefore, saving computational resources and yielding a positive impact on the environment. Moreover, our method can help smaller research organizations with limited access to resources to be competitive in the deep learning domain, which reduces the investment costs on hardware. Although our method significantly reduces the time taken for optimizing a machine learning algorithm that achieves peak performance, we warn against running our method for an extended time only to achieve marginal gains in performance, unless it is mission-critical. Last but not least, in order to save energy, we invite the community to create sparse benchmarks with surrogates, instead of dense tabular ones.

A Experimental Setup

A.1 Benchmarks

LCBench. LCBench[4] is a feedforward neural network benchmark on tabular data which consists of 2000 configuration settings for each of the 35 datasets. The configurations were evaluated during HPO runs with AutoPyTorch. LCBench features a search space of 7 numerical hyperparameters, where every hyperparameter configuration is trained for 50 epochs. The objective is to optimize seven different hyperparameters of funnel-shaped neural networks, i.e., batch size, learning rate, momentum, weight decay, dropout, number of layers, and maximum number of units per layer.

TaskSet. TaskSet[5] is a benchmark that features over 1162 diverse tasks from different domains and includes 5 search spaces. In this work, we focus on NLP tasks and we use the Adam8p search space with 8 continuous hyperparameters. We refer to Figure 11 for the exact task names considered in our experiments. The learning curves provided in TaskSet report scores after every 200 iterations. We refer to those as "steps". The objective is to optimize eight hyperparameters for a set of different recurrent neural networks (RNN) that differ in embedding size, RNN cell, and other architectural features. The set of hyperparameters consists of optimizer-specific hyperparameters, such as the learning rate, the exponential decay rate of the first and second momentum of Adam, β_1 and β_2, and Adam's constant for numerical stability ε. Furthermore, there are two hyperparameters controlling linear and exponential learning rate decays, as well as L1 and L2 regularization terms.

NAS-Bench-201. NAS-Bench-201[6] is a benchmark that has precomputed about 15,600 architectures trained for 200 epochs for the image classification datasets CIFAR-10, CIFAR-100, and ImageNet. The objective is to select for each of the six operations within the cell of the macro architecture one of five different operations. All other hyperparameters such as learning rate and batch size are kept fixed. NAS-Bench-201 features a search space of 6 categorical hyperparameters and each architecture is trained for 200 epochs.

A.2 Preprocessing

In the following, we describe the preprocessing applied to the hyperparameter representation. For LCBench, we apply a log-transform to batch size, learning rate, and weight decay. For TaskSet, we apply it on the learning rate, L1 and L2 regularization terms, linear and exponential decay of the learning rate. All continuous hyperparameters are scaled to the range between 0 and 1 using sklearn's MinMaxScaler. If not mentioned otherwise, we use one-hot encoding for the categorical hyperparameters. As detailed in subsection A.5, some baselines have a specific way of dealing with them. In that case, we use the method recommended by the authors.

A.3 Framework

The framework contains the evaluated hyperparameters and their corresponding validation curves. The list of candidate hyperparameters is passed to the baseline-specific interface, which in turn,

[4] `https://github.com/automl/LCBench`

[5] `https://github.com/google-research/google-research/tree/master/task_set`

[6] `https://github.com/D-X-Y/NAS-Bench-201`

optimizes and queries the framework for the hyperparameter configuration that maximizes utility. Our framework in turn responds with the validation curve and the cost of the evaluation. In case a hyperparameter configuration has been evaluated previously up to a budget b and a baseline requires the response for budget $b + 1$, the cost is calculated accordingly only for the extra budget requested.

A.4 Implementation Details

We implement the Deep Kernel Gaussian Process using GPyTorch 1.5 [Gardner et al., 2018]. We use an RBF kernel and the dense layers of the transformation function φ have 128 and 256 units. We used a convolutional layer with a kernel size of three and four filters. All parameters of the Deep Kernel are estimated by maximizing the marginal likelihood. We achieve this by using gradient ascent and Adam [Kingma and Ba, 2015] with a learning rate of 0.1 and batch size of 64. We stop training as soon as the training likelihood is not improving for 10 epochs in a row or we completed 1,000 epochs. For every new data point, we start training the GP with its old parameters to reduce the required effort for training.

A.5 Baselines

Random Search & Hyperband. Random search and Hyperband sample hyperparameter configurations at random and therefore the preprocessing is irrelevant. We have implemented both from scratch and use the recommended hyperparameters for Hyperband, i.e. $\eta = 3$.

BOHB. For our experiments with BOHB, we use version 0.7.4 of the officially-released code[7].

DEHB. For our experiments with DEHB, we use the official public implementation[8]. We developed an interface that communicates between our framework and DEHB. In addition to the initial preprocessing common for all methods, we encode categorical hyperparameters with a numerical value in the interval [0, 1]. For a categorical hyperparameter $\mathbf{x}_i$, we take N_i equal-sized intervals, where N_i represents the number of unique categorical values for hyperparameter $\mathbf{x}_i$ and we assign the value for a categorical value $n \in N_i$ to the middle of the interval $[n, n + 1]$ as suggested by the authors. For configuring the DEHB algorithm we used the default values from the library.

Dragonfly. We use the publicly available code of Dragonfly[9]. No special treatment of categorical hyperparameters is required since Dragonfly has its own way to deal with them. We use version $0.1.6$ with default settings.

MF-DNN. We use the official implementation of MF-DNN by the authors[10]. Initially, we tried to use multiple incremental fidelity levels like for DyHPO, however, the method runtime was too high and it could not achieve competitive results. For that reason, we use only a few fidelity levels like the authors do in their work Li et al. [2020b]. We use the same fidelity levels as for Hyperband, DEHB, and BOHB to have a fair comparison between the baselines. We also use the same number of initial points as for the other methods to have the same maximal resource allocated for every fidelity level.

ASHA-HB. We use the public implementation from the well-known optuna library (version $2.10.0$). We used the same eta, minimum and maximal budget as for HB, DEHB, and BOHB in our experiments, to have a fair comparison.

B Additional Plots

In Figure 8, we ablate the learning curve input in our kernel, to see the effect it has on performance for the CIFAR-10 and CIFAR-100 datasets from the NAS-Bench-201 benchmark. The results indicate that the learning curve plays an important role in achieving better results by allowing faster convergence and a better anytime performance.

[7] `https://github.com/automl/HpBandSter`

[8] `https://github.com/automl/DEHB/`

[9] `https://github.com/dragonfly/dragonfly`

[10] `https://github.com/shib0li/DNN-MFBO`

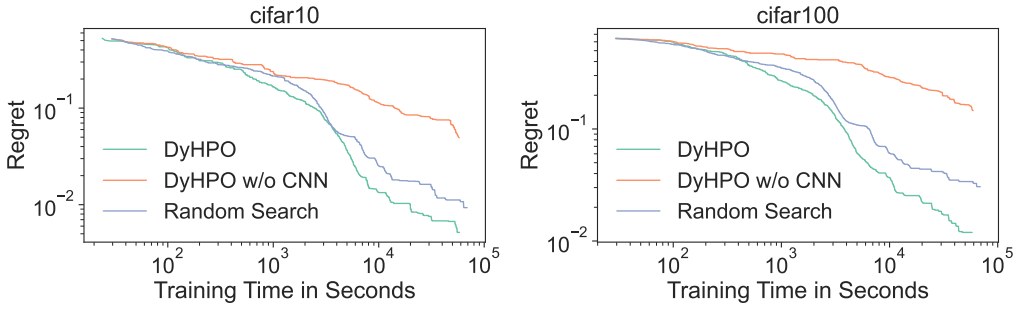

Figure 8: The learning curve ablation for the CIFAR-10 and CIFAR-100 tasks of NAS-Bench-201.

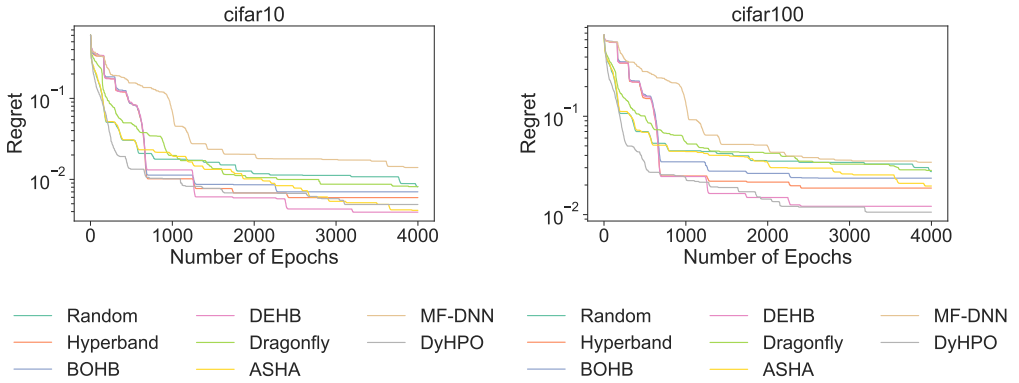

Figure 9: NAS-Bench-201 regret results over the number of epochs spent during the optimization.

Additionally, in Figure 9, we show the performance comparison over the number of epochs of every method for the CIFAR-10 and CIFAR-100 datasets in the NAS-Bench-201 benchmark. While, in Figure 10, we present the performance comparison over time. As can be seen, DyHPO converges faster and has a better performance compared to the other methods over the majority of the time or steps, however, towards the end although it is the optimal method or close to the optimal method, the difference in regret is not significant anymore.

Furthermore, Figure 11 shows the performance comparison for the datasets chosen from TaskSet over the number of steps. Looking at the results, DyHPO is outperforming all methods convincingly on the majority of datasets by converging faster and with significant differences in the regret evaluation metric.

In Figure 12 and 13, we show the performance comparison for all the datasets from LCBench regarding regret over the number of epochs. Similarly, in Figure 14 and 15, we show the same performance comparison, however, over time. As can be seen, DyHPO manages to outperform the other competitors in the majority of the datasets, and in the datasets that it does not, it is always close to the top-performing method, and the difference between methods is marginal.

In Figure 16 we provide the extended results of **Experiment 3** for TaskSet. We show the precision, average regret, and promotion percentage for poor-performing configurations for DyHPO and the other competitor methods.

Lastly, we explore the behavior of DyHPO after finding the configuration which is returned at the end of the optimization as the best configuration. In Figure 17, we show how the budget is distributed on the configurations considered during that part of the optimization. Clearly, DyHPO is spending very little budget on most configurations. Furthermore, we investigated how many new configurations are considered during this phase. For LCBench, 76.98% of considered configurations are new demonstrating that DyHPO is investigating most of the budget into exploration. These are even more extreme for TaskSet (93.16% and NAS-Bench-201 (97.51%).

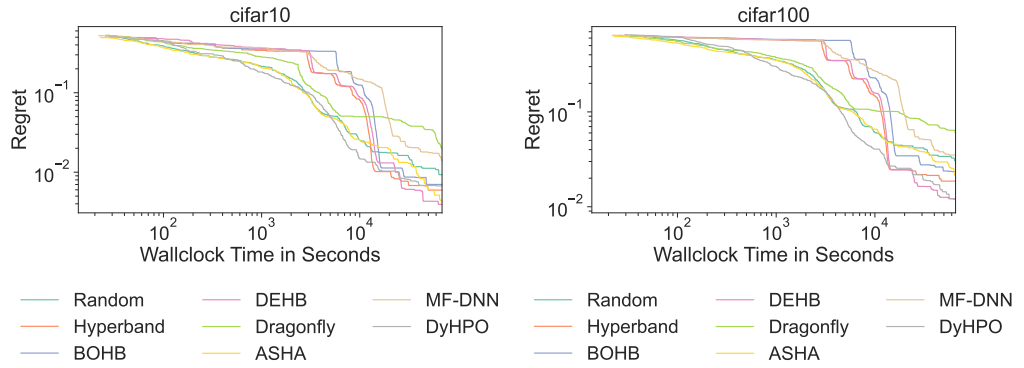

Figure 10: NAS-Bench-201 regret results over the total optimization time. The total time includes the method overhead time and the hyperparameter configuration evaluation time.

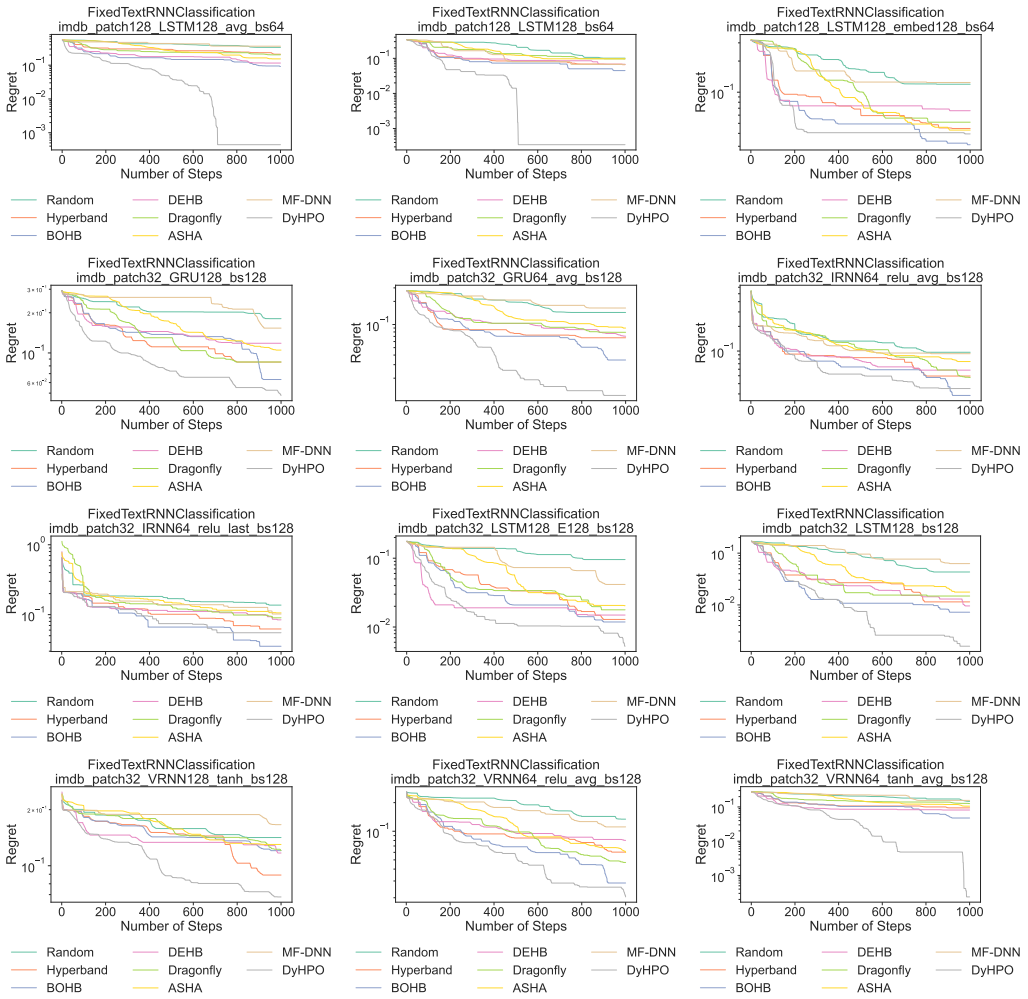

Figure 11: Performance comparison over the number of steps on a dataset level for TaskSet.

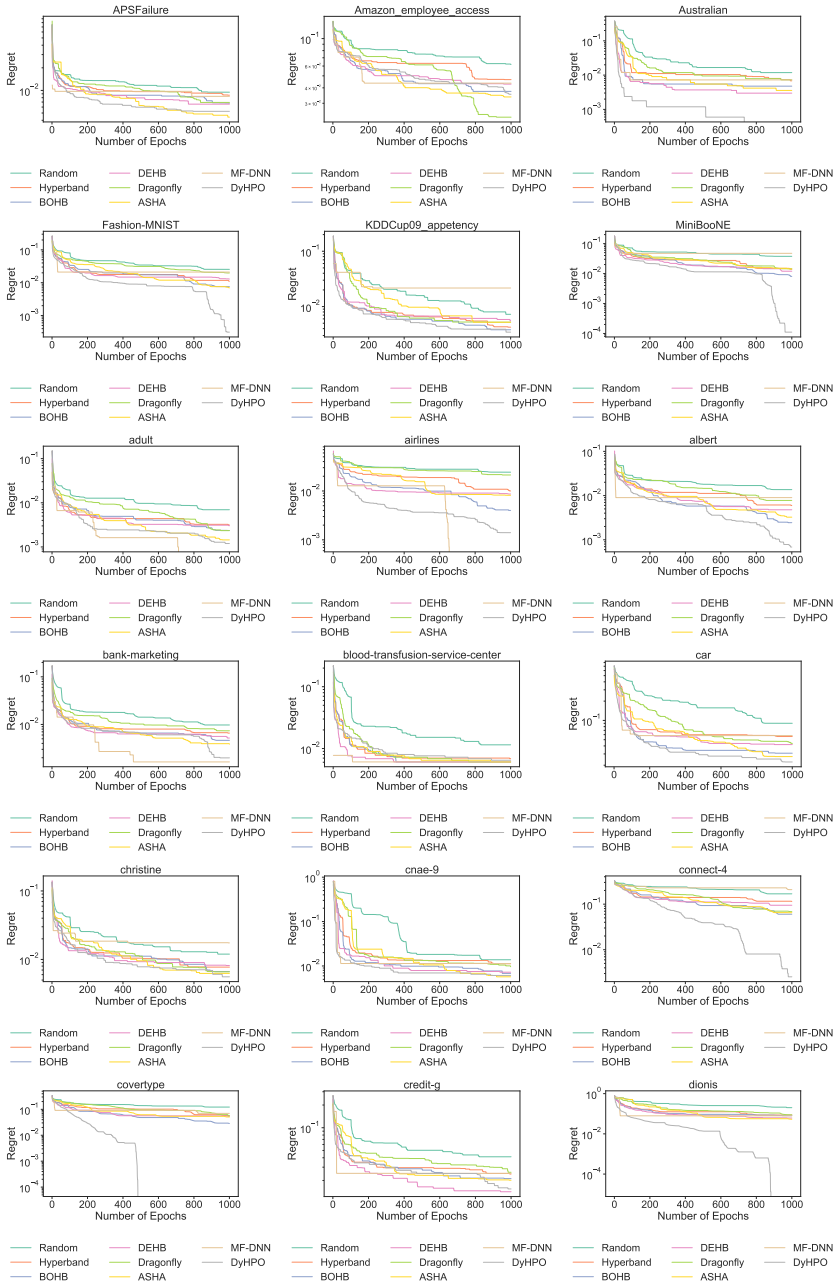

Figure 12: Performance comparison over the number of steps on a dataset level for LCBench.

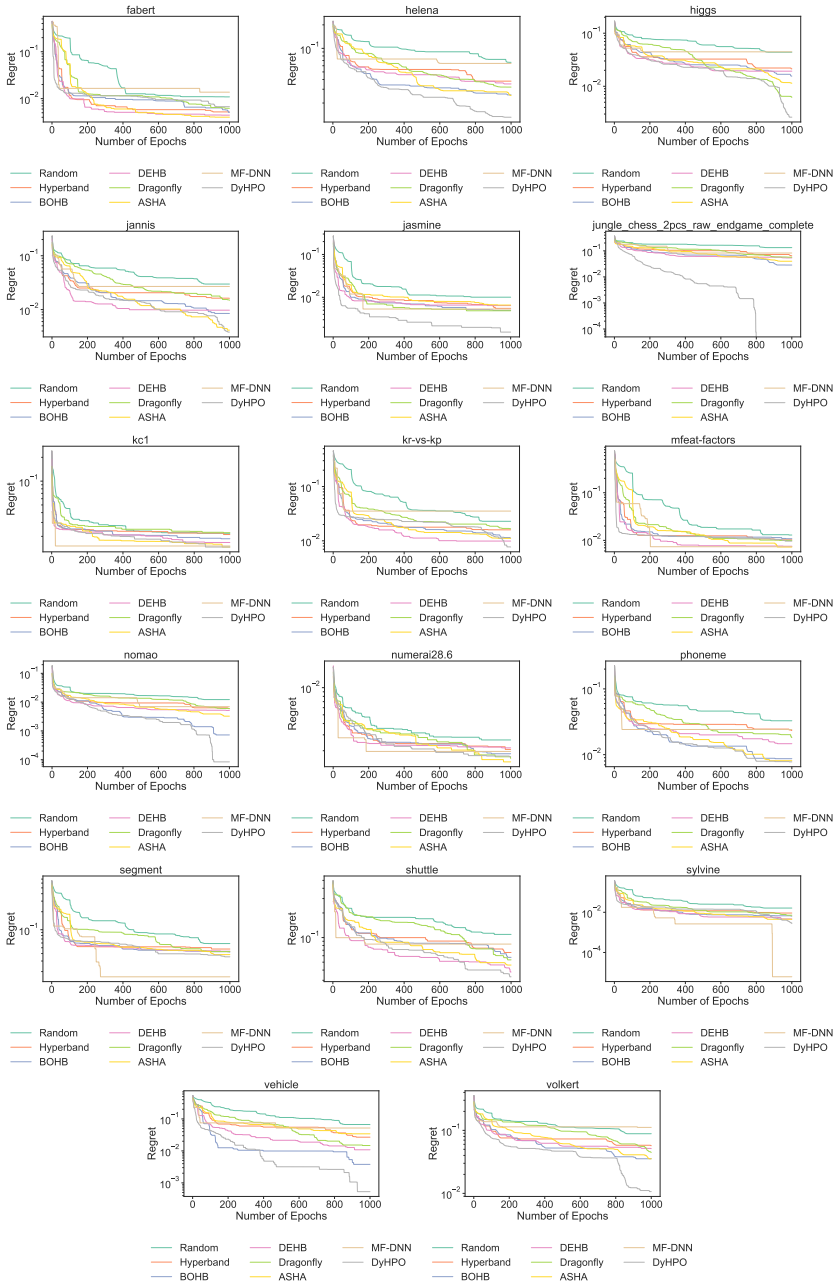

Figure 13: Performance comparison over the number of steps on a dataset level for LCBench (cont.).

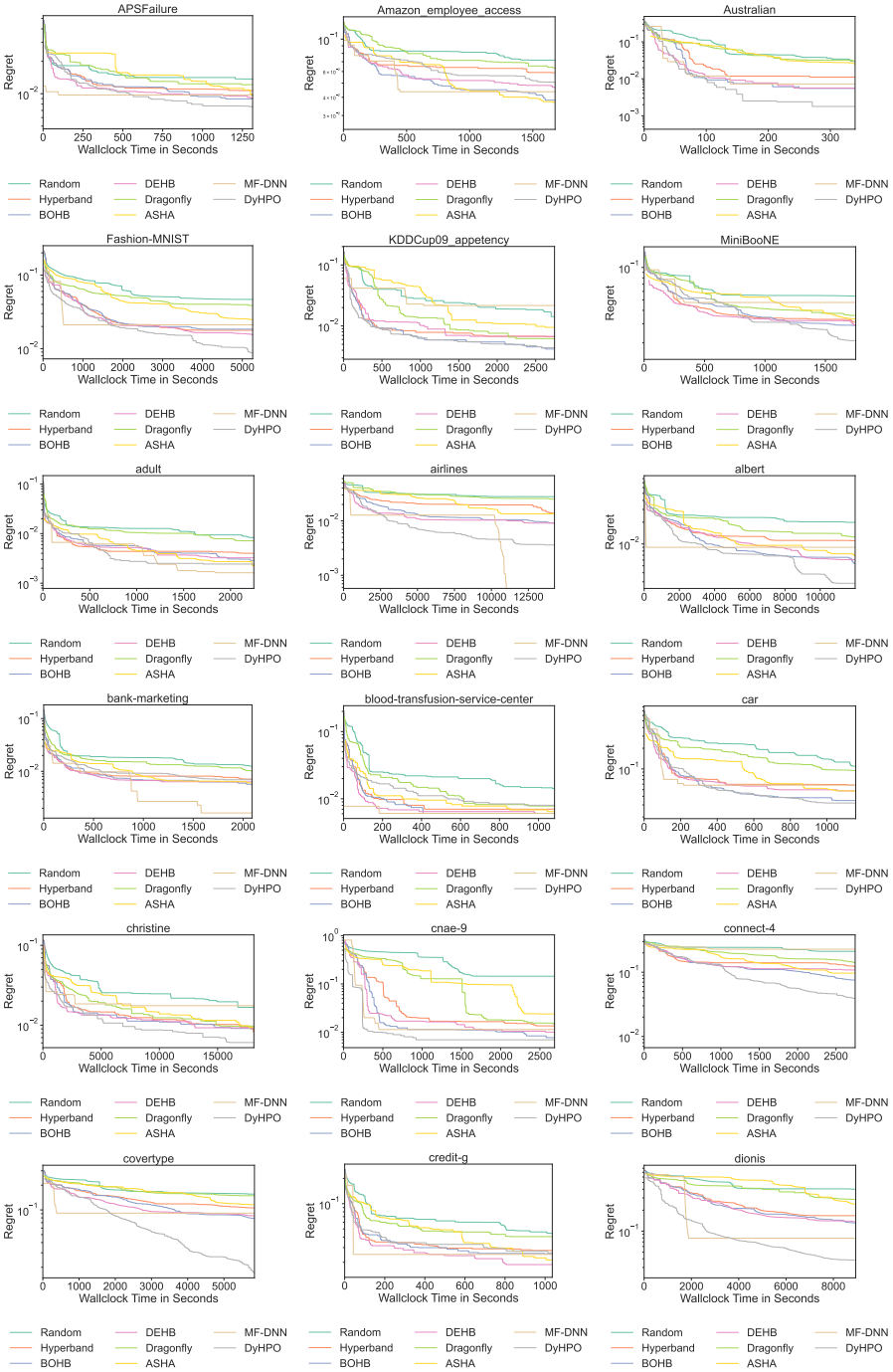

Figure 14: Performance comparison over time on a dataset level for LCBench with the overhead included.

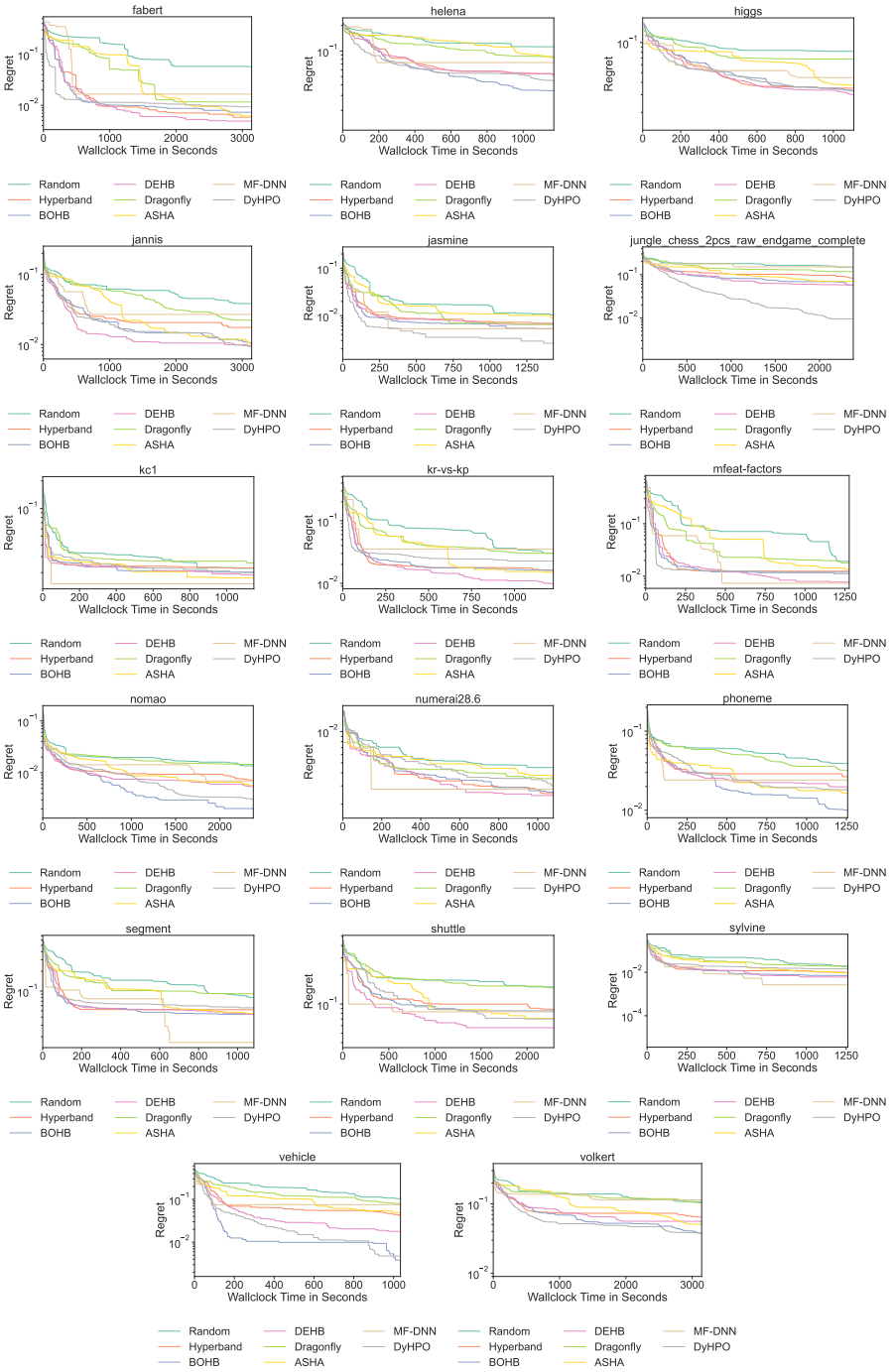

Figure 15: Performance comparison over time on a dataset level for LCBench with the overhead included. (cont.).

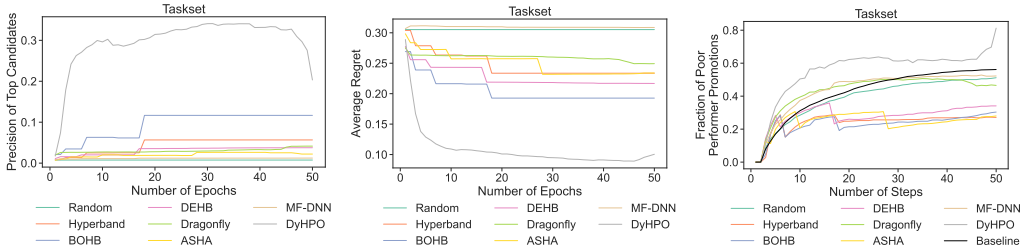

Figure 16: The efficiency of DyHPO as the optimization progresses. **Left:** The fraction of top-performing candidates from all candidates that were selected to be trained. **Middle:** The average regret for the configurations that were selected to be trained at a given budget. **Right:** The percentage of configurations that belong to the top 1/3 configurations at a given budget and that were in the top bottom 2/3 of the configurations at a previous budget. All of the results are from the Taskset benchmark.

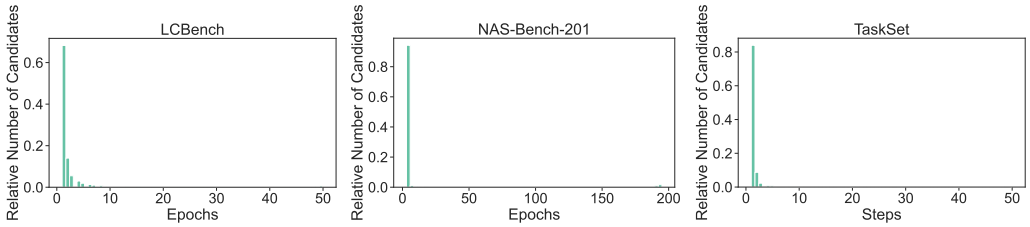

Figure 17: These plots shed light on how DyHPO behaves *after* the configuration it finally returns as the best. The plots show how many epochs are spent per candidate. As we can see, for most candidates only a small budget was considered, indicating that DyHPO is mostly exploring at this point.