Dynamic and Efficient Gray-Box Hyperparameter Optimization for Deep Learning

Martin Wistuba[1] **Arlind Kadra**[2] **Josif Grabocka**[2]

[1]Amazon Search, Berlin, Germany
[2]University of Freiburg, Freiburg, Germany

Abstract Gray-box hyperparameter optimization techniques have recently emerged as a promising direction for tuning Deep Learning methods. In this work, we introduce DyHPO, a method that learns to dynamically decide which configuration to try next, and for what budget. Our technique is a modification to the classical Bayesian optimization for a gray-box setup. Concretely, we propose a new surrogate for Gaussian Processes that embeds the learning curve dynamics and a new acquisition function that incorporates multi-budget information. We demonstrate the significant superiority of DyHPO against state-of-the-art hyperparameter optimization baselines through large-scale experiments comprising 50 datasets (Tabular, Image, NLP) and diverse neural networks (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 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 runs random configurations at different budgets (e.g. different 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 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

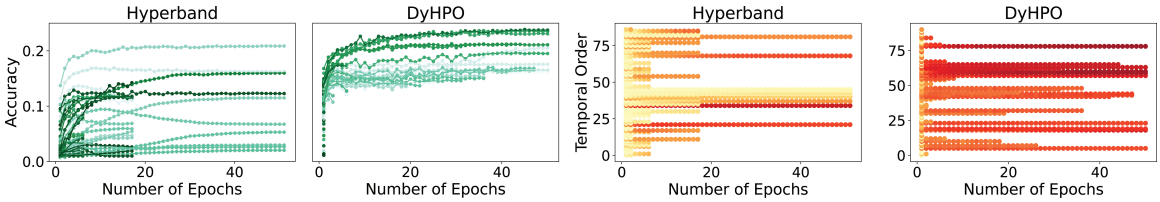


Figure 1: Green: Learning curves observed during the search. The darker the learning curve, the later it was evaluated during the search. Red: y-axis shows a sequence of learning curve evaluations (bottom to top). The color indicates accuracy. The darker red the higher the accuracy.

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 gray-box method that *dynamically* decides how many configurations to try and how much budget to spend on each configuration. DyHPO is a Bayesian Optimization (BO) approach based on Gaussian Processes (GP), that extends and adapts BO to the multi-fidelity case. 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. We illustrate the differences between our strategy and successive halving with the experiment of Figure 1, where we showcase the HPO progress. Hyperband *statically* pre-allocates the budget for a set of candidates according to a predefined policy. However, DyHPO *dynamically* adapts the allocation of budgets for configurations after every HPO step.

The two plots on the left of Figure 1 show the learning curves of multiple neural networks trained with different hyperparameter configurations. The darker the color of a learning curve, the later the model corresponding to the learning curve was trained during the optimization process. In an optimal scenario, there should be no learning curve of darker color trained for a longer time if there is already a learning curve of lighter color with higher accuracy. We see that this is not the case for Hyperband since it is possible that no configuration selected for a Hyperband bracket is outperforming the current best configuration. A dynamic budget allocation as proposed for our method DyHPO overcomes this issue. We see that it invests only a small budget into configurations that show little promise.

2 Dynamic Multi-Fidelity HPO

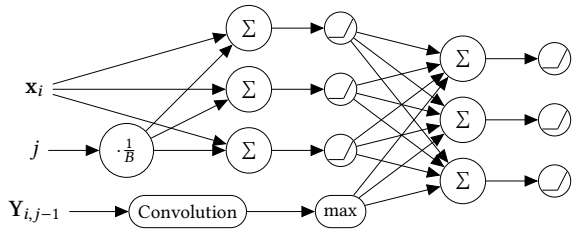
Notation. The gray-box HPO setting allows querying configurations with a smaller budget compared to the total maximal budget B. Thus, we can query from the response function $f : \mathcal{X} \times \mathbb{N} \to \mathbb{R}$ where $f_{i,j} = f(\mathbf{x}_i, j)$ is the response after spending a budget 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 $\varepsilon_j \sim \mathcal{N}(0, \sigma^2_{j,n})$. We denote the learning curve as $\mathbf{Y}_{i,j-1} = (y_{i,1}, \ldots, y_{i,j-1})$.

We discuss Gaussian processes (GP) in detail in Appendix A but we assume the reader has a basic understanding. In the following, we will refer to its kernel function as k.

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 (Wilson et al., 2016) as:

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

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



(a) The feature extractor φ used in our deep kernel.

DyHPO Algorithm

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

(b) DyHPO Algorithm

Figure 2: Deep kernel architecture and DyHPO algorithm.

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.

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:

$$\hat{\theta}, \hat{w} = \arg\max_{\theta,w} p(\mathbf{y}|\mathbf{X}, \mathbf{Y}, \theta, w) \propto \arg\min_{\theta,w} \mathbf{y}^{\mathrm{T}}\mathbf{K}(\theta, w)^{-1}\mathbf{y} + \log|\mathbf{K}(\theta, w)| \tag{2}$$

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 $p\left(f_{i,j}|\mathbf{x}_i, \mathbf{Y}_{i,j-1}, j, \mathcal{D}, \hat{\boldsymbol{\theta}}, \hat{\mathbf{w}}\right)$ as detailed in Appendix A.

Multi-Fidelity Expected Improvement. Expected improvement (Jones et al., 1998) is a commonly used acquisition function and is defined as:

$$\mathrm{EI}(\mathbf{x}|\mathcal{D}) = \mathbb{E}\left[\max\left\{f(\mathbf{x}) - y^{\max}, 0\right\}\right], \tag{3}$$

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

$$\mathrm{EI}_{\mathrm{MF}}(\mathbf{x}, j|\mathcal{D}) = \mathbb{E}\left[\max\left\{f(\mathbf{x}, j) - y_j^{\max}, 0\right\}\right], \tag{4}$$

where:

$$y_j^{\max} = \begin{cases} \max\{y \mid ((\mathbf{x}, \cdot, j), y) \in \mathcal{D}\} & \text{if } ((\mathbf{x}, \cdot, j), y) \in \mathcal{D} \\ \max\{y \mid (\cdot, y) \in \mathcal{D}\} & \text{otherwise} \end{cases} \tag{5}$$

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.

The DyHPO Algorithm. The DyHPO algorithm looks very similar to many black-box Bayesian optimization algorithms as shown in Figure 2b. The big difference is that at each step we dynamically decide which candidate configuration to train *for a small additional budget.* 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 our previously described acquisition function 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 j such that it is by exactly one higher than the budget used to evaluate $\mathbf{x}_i$ before. If $\mathbf{x}_i$ has not been evaluated for any budget yet, j is set to 1. 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 Equation (2).

3 Experimental Protocol

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. In our experiments, we report the mean of ten repetitions and we report two common metrics such as 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 C.4.

Baselines. In our experiments, we compare against well-known baselines such as Random Search (Bergstra and Bengio, 2012), Hyperband (Li et al., 2017), BOHB (Falkner et al., 2018), DEHB (Awad et al., 2021), and ASHA (Li et al., 2020a). Furthermore, we compare against methods more closely related to ours. We compare against MF-DNN, a multi-fidelity Bayesian optimization method that uses deep neural networks to capture the relationships between different fidelities Li et al. (2020b). Furthermore, 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.

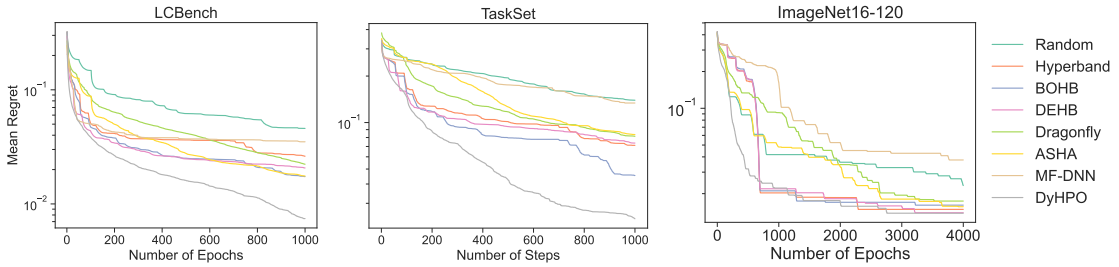


Figure 3: The mean regret for the different benchmarks over the number of epochs or steps (every 200 iterations). The results are aggregated over 35 different datasets for LCBench and aggregated over 12 different NLP tasks for TaskSet.

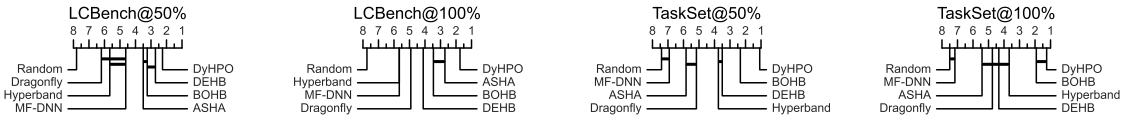


Figure 4: Critical difference diagram for LCBench and TaskSet. The results correspond to results after 500 and 1,000 epochs. Connected ranks via a bold bar indicate that performances are not significantly different ($p > 0.05$).

Results. In our first experiment, we evaluate the various methods on LCBench (Zimmer et al., 2021) (neural architecture optimization for tabular datasets), TaskSet (Metz et al., 2020) (hyperparameter optimization for NLP models), and NAS-Bench-201 (Dong and Yang, 2020) (neural architecture search for image data). We provide more details about the benchmarks in Appendix C.1.

We show the aggregated results on those benchmarks in Figure 3. 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 and to the performance comparison over time, we refer the reader to Appendix D.

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 number 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.

4 Broader Impact and Limitations

Automatically tuning the hyper-parameters of Deep Learning systems opens a garden of delights in terms of making AI plug-and-play, and minimizing the dependency on expensive/scarce AI experts. Furthermore, efficient HPO algorithms like ours significantly reduce the energy consumption footprint, because they can find good-performing configurations with less HPO runtime/compute. Given that configuring and deploying Deep Learning is an acute and open challenge for virtually all application domains, we expect our method to have a broad impact. As a non-critical limitation, our method is currently evaluated on several search spaces with a large but finite amount of candidates. This is not truly reflecting the nature of continuous search spaces which may or may not add to the complexity of the problem and require changes to our method.

5 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.

References

Awad, N. H., Mallik, N., and Hutter, F. (2021). 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.

Bergstra, J., Bardenet, R., Bengio, Y., and Kégl, B. (2011). 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.

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13:281–305.

Bertrand, H., Ardon, R., Perrot, M., and Bloch, I. (2017). Hyperparameter optimization of deep neural networks: Combining hyperband with bayesian model selection. In *Conférence sur l'Apprentissage Automatique*.

Chen, Y., Hoffman, M. W., Colmenarejo, S. G., Denil, M., Lillicrap, T. P., Botvinick, M., and de Freitas, N. (2017). 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.

Demsar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.*, 7:1–30.

Dong, X. and Yang, Y. (2020). 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*.

Falkner, S., Klein, A., and Hutter, F. (2018). 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.

Franceschi, L., Donini, M., Frasconi, P., and Pontil, M. (2017). 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.

Gardner, J. R., Pleiss, G., Weinberger, K. Q., Bindel, D., and Wilson, A. G. (2018). 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.

Jamieson, K. G. and Talwalkar, A. (2016). 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.

Jones, D. R., Schonlau, M., and Welch, W. J. (1998). Efficient global optimization of expensive black-box functions. *J. Global Optimization*, 13(4):455–492.

Kandasamy, K., Dasarathy, G., Oliva, J. B., Schneider, J. G., and Póczos, B. (2016). 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.

Kandasamy, K., Dasarathy, G., Schneider, J. G., and Póczos, B. (2017). 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.

Kandasamy, K., Neiswanger, W., Schneider, J., Póczos, B., and Xing, E. P. (2018). 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.

Kandasamy, K., Vysyaraju, K. R., Neiswanger, W., Paria, B., Collins, C. R., Schneider, J., Póczos, B., and Xing, E. P. (2020). Tuning hyperparameters without grad students: Scalable and robust bayesian optimisation with dragonfly. *J. Mach. Learn. Res.*, 21:81:1–81:27.

Kingma, D. P. and Ba, J. (2015). 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*.

Klein, A., Falkner, S., Bartels, S., Hennig, P., and Hutter, F. (2017). 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.

Li, L., Jamieson, K., Rostamizadeh, A., Gonina, E., Ben-Tzur, J., Hardt, M., Recht, B., and Talwalkar, A. (2020a). A system for massively parallel hyperparameter tuning. *Proceedings of Machine Learning and Systems*, 2:230–246.

Li, L., Jamieson, K. G., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. (2017). Hyperband: A novel bandit-based approach to hyperparameter optimization. *J. Mach. Learn. Res.*, 18:185:1–185:52.

Li, S., Xing, W., Kirby, R. M., and Zhe, S. (2020b). 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*.

Lorraine, J., Vicol, P., and Duvenaud, D. (2020). 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.

Maclaurin, D., Duvenaud, D., and Adams, R. P. (2015). 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.

Metz, L., Maheswaranathan, N., Sun, R., Freeman, C. D., Poole, B., and Sohl-Dickstein, J. (2020). Using a thousand optimization tasks to learn hyperparameter search strategies. *CoRR*, abs/2002.11887.

Parker-Holder, J., Nguyen, V., and Roberts, S. J. (2020). 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*.

Perrone, V., Jenatton, R., Seeger, M. W., and Archambeau, C. (2018). 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.

Poloczek, M., Wang, J., and Frazier, P. I. (2017). 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.

Rai, A., Desai, R., and Goyal, S. (2016). Bayesian optimization with a neural network kernel.

Snoek, J., Larochelle, H., and Adams, R. P. (2012). 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.

Srinivas, N., Krause, A., Kakade, S. M., and Seeger, M. W. (2010). 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.

Swersky, K., Snoek, J., and Adams, R. P. (2013). 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.

Swersky, K., Snoek, J., and Adams, R. P. (2014). Freeze-thaw bayesian optimization. *CoRR*, abs/1406.3896.

Takeno, S., Fukuoka, H., Tsukada, Y., Koyama, T., Shiga, M., Takeuchi, I., and Karasuyama, M. (2020). 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.

Wang, J., Xu, J., and Wang, X. (2018). Combination of hyperband and bayesian optimization for hyperparameter optimization in deep learning. *CoRR*, abs/1801.01596.

Wilson, A. G., Hu, Z., Salakhutdinov, R., and Xing, E. P. (2016). 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.

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

Wistuba, M. and Grabocka, J. (2021). Few-shot bayesian optimization with deep kernel surrogates. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*.

Zimmer, L., Lindauer, M., and Hutter, F. (2021). Auto-pytorch: Multi-fidelity metalearning for efficient and robust autodl. *IEEE Trans. Pattern Anal. Mach. Intell.*, 43(9):3079–3090.

A Gaussian Processes

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:

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(m\left(\mathbf{X}, \mathbf{x}_*\right), \begin{pmatrix} \mathbf{K}_n & \mathbf{K}_* \\ \mathbf{K}_*^T & \mathbf{K}_{**} \end{pmatrix}\right) \quad . \tag{6}$$

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:

$$\mathbb{E}\left[\mathbf{f}_*|\mathbf{X}, \mathbf{y}, \mathbf{X}_*\right] = \mathbf{K}_*^T\mathbf{K}_n^{-1}\mathbf{y} \qquad \text{cov}\left[\mathbf{f}_*|\mathbf{X}, \mathbf{X}_*\right] = \mathbf{K}_{**} - \mathbf{K}_*^T\mathbf{K}_n^{-1}\mathbf{K}_* \tag{7}$$

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 $\mathbf{K}\left(\theta, w\right) \coloneqq k(\varphi(\mathbf{x}, w), \varphi(\mathbf{x}'; w)|\boldsymbol{\theta})$, which allows us to use convolutional operations as part of our kernel.

B Detailed Discussion of Related Work

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., 2017), 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 discrete and large hyperparameter search spaces better than BOHB.

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 the first deep kernel GP for multi-fidelity HPO that is able to capture the learning dynamics across fidelities/budgets, combined with an acquisition function that is tailored for the gray-box setup.

C Experimental Setup

C.1 Benchmarks

LCBench. LCBench[1] 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[2] 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 9 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[3] 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.

C.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 to 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 C.5, some baselines have a specific way of dealing with them. In that case, we use the method recommended by the authors.

[1] https://github.com/automl/LCBench

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

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

C.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, 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.

C.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.

C.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[4].

DEHB. For our experiments with DEHB, we use the official public implementation[5]. 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[6]. 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[7]. 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.

[4] https://github.com/automl/HpBandSter

[5] https://github.com/automl/DEHB/

[6] https://github.com/dragonfly/dragonfly

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

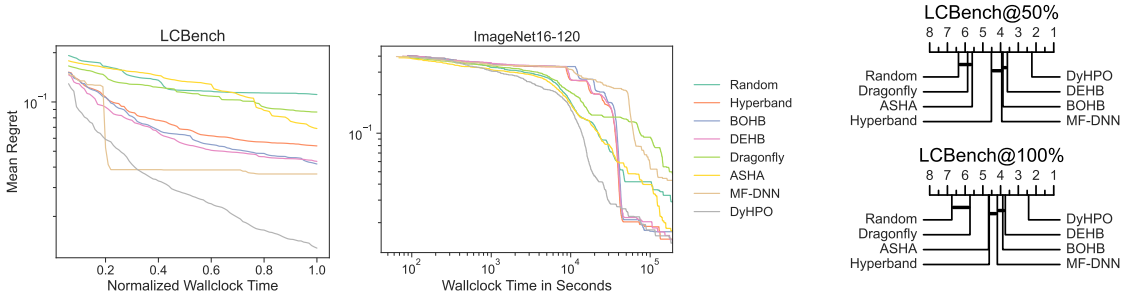


Figure 5: **Left**: The regret over time for all methods during the optimization procedure for the LCBench benchmark and the ImageNet dataset from the NAS-Bench-201 benchmark. The normalized wallclock time represents the actual runtime divided by the total wallclock time of DyHPO including the overhead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

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.

D Additional Plots

D.1 Considering Overhead

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 the next subsection.

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.

D.2 Further Insights

In Figure 6, 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.

Additionally, in Figure 7, 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 8, 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 9 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 10 and 11, we show the performance comparison for all the datasets from LCBench regarding regret over the number of epochs. Similarly, in Figure 12 and 13, 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.

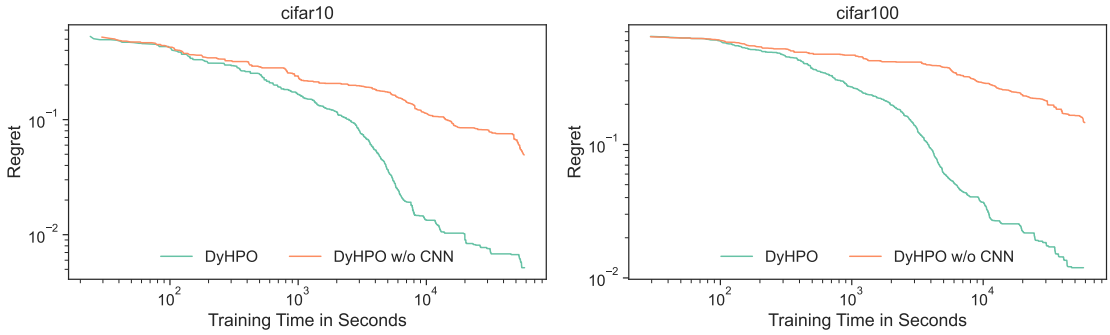


Figure 6: The learning curve ablation for the CIFAR-10 and CIFAR-100 tasks of NAS-Bench-201.

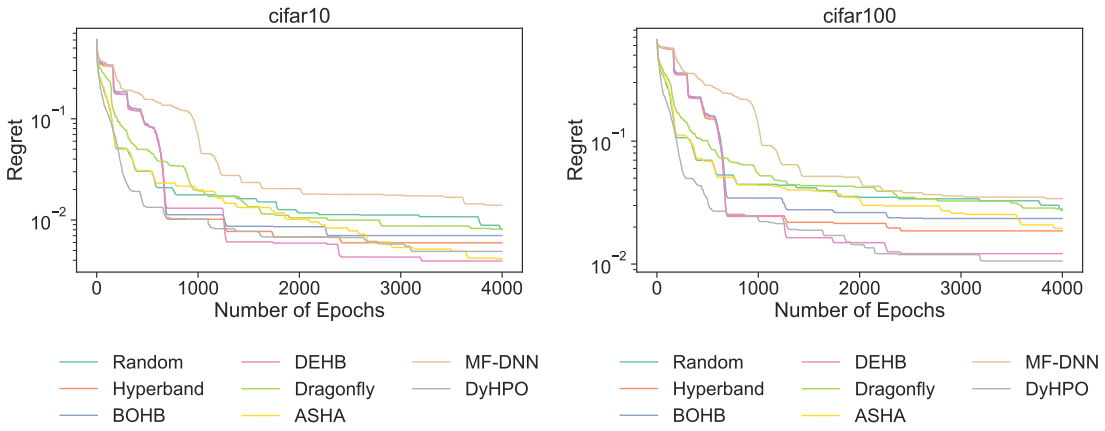


Figure 7: NAS-Bench-201 regret results over the number of epochs spent during the optimization.

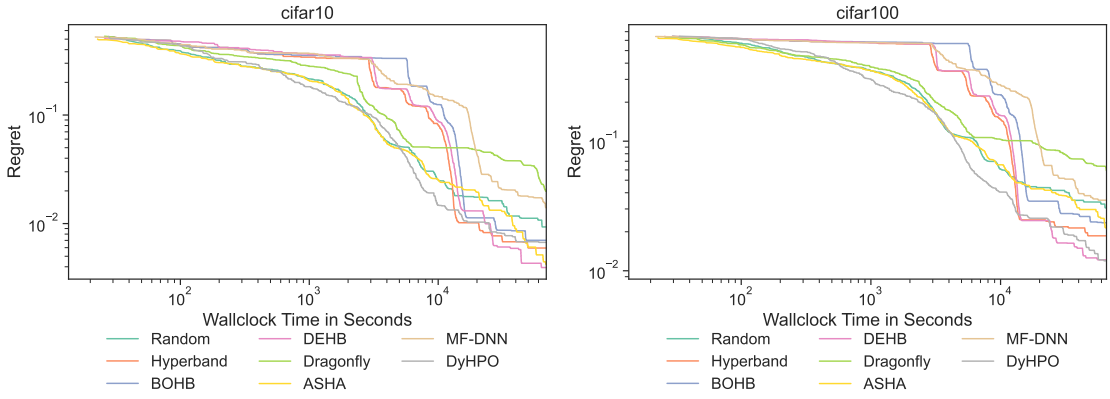


Figure 8: NAS-Bench-201 regret results over the total optimization time. The total time includes the method overhead time and the hyperparameter configuration evaluation time.

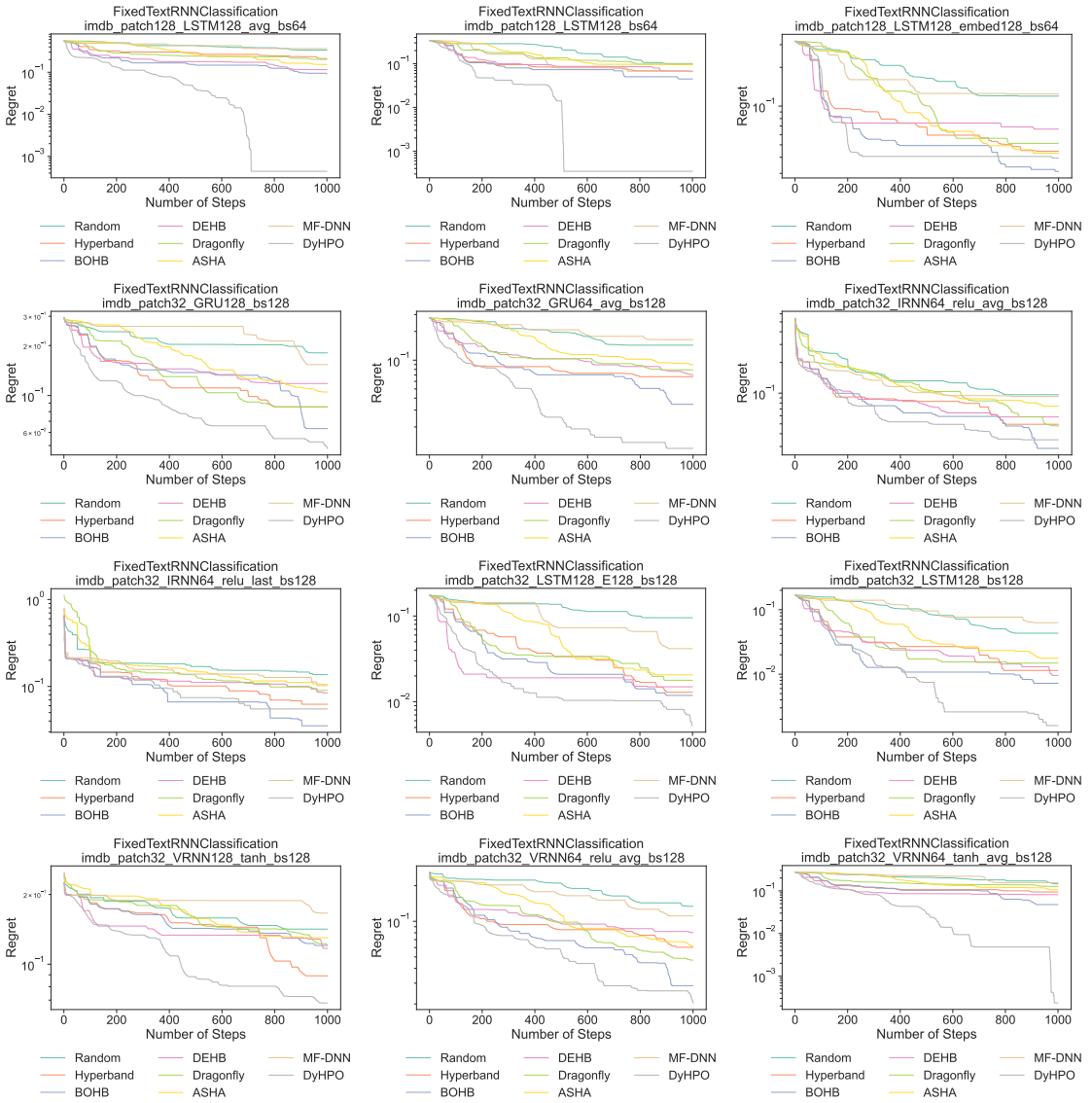


Figure 9: Performance comparison over the number of steps on a dataset level for TaskSet.

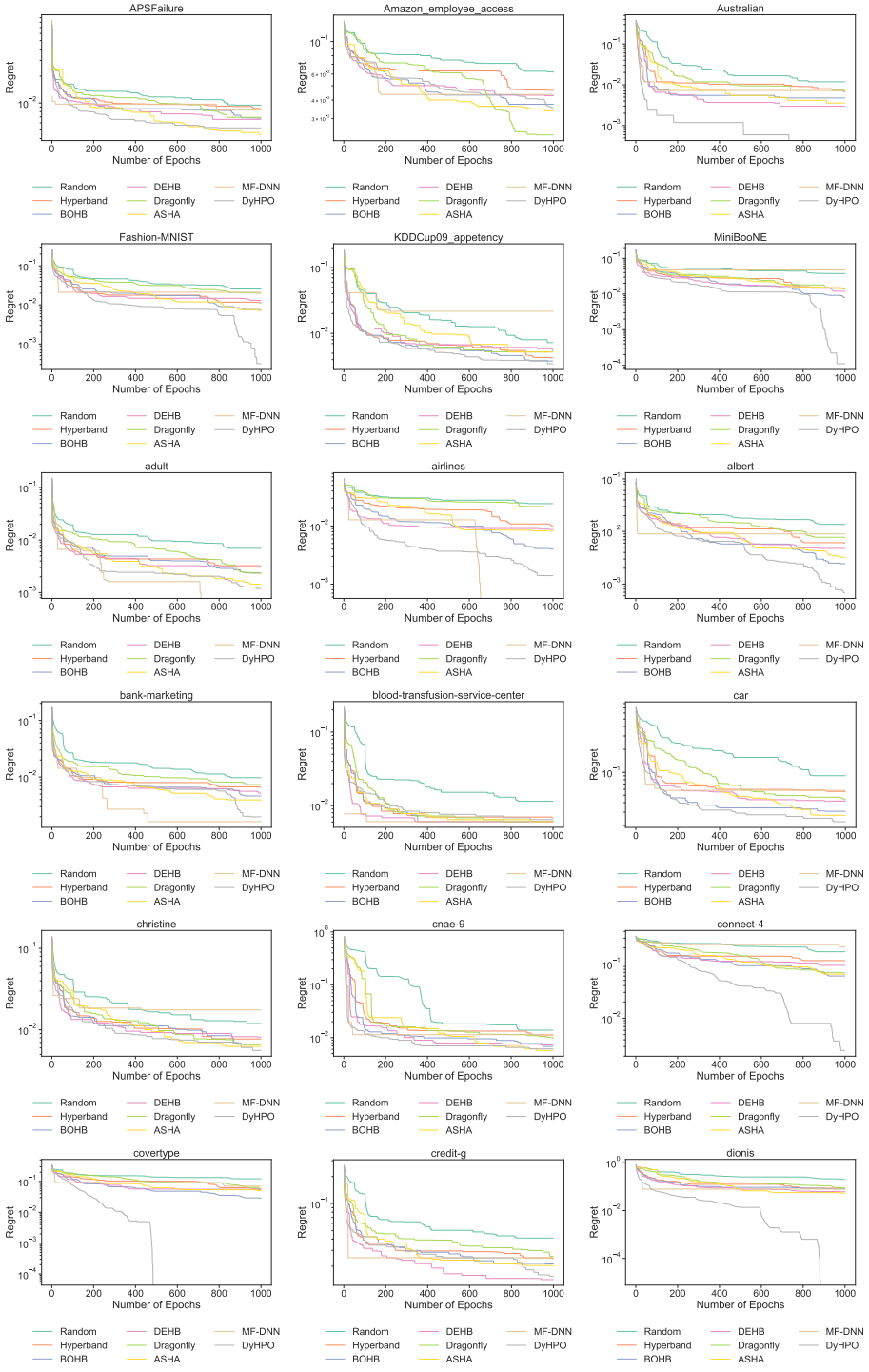


Figure 10: Performance comparison over the number of steps on a dataset level for LCBench.

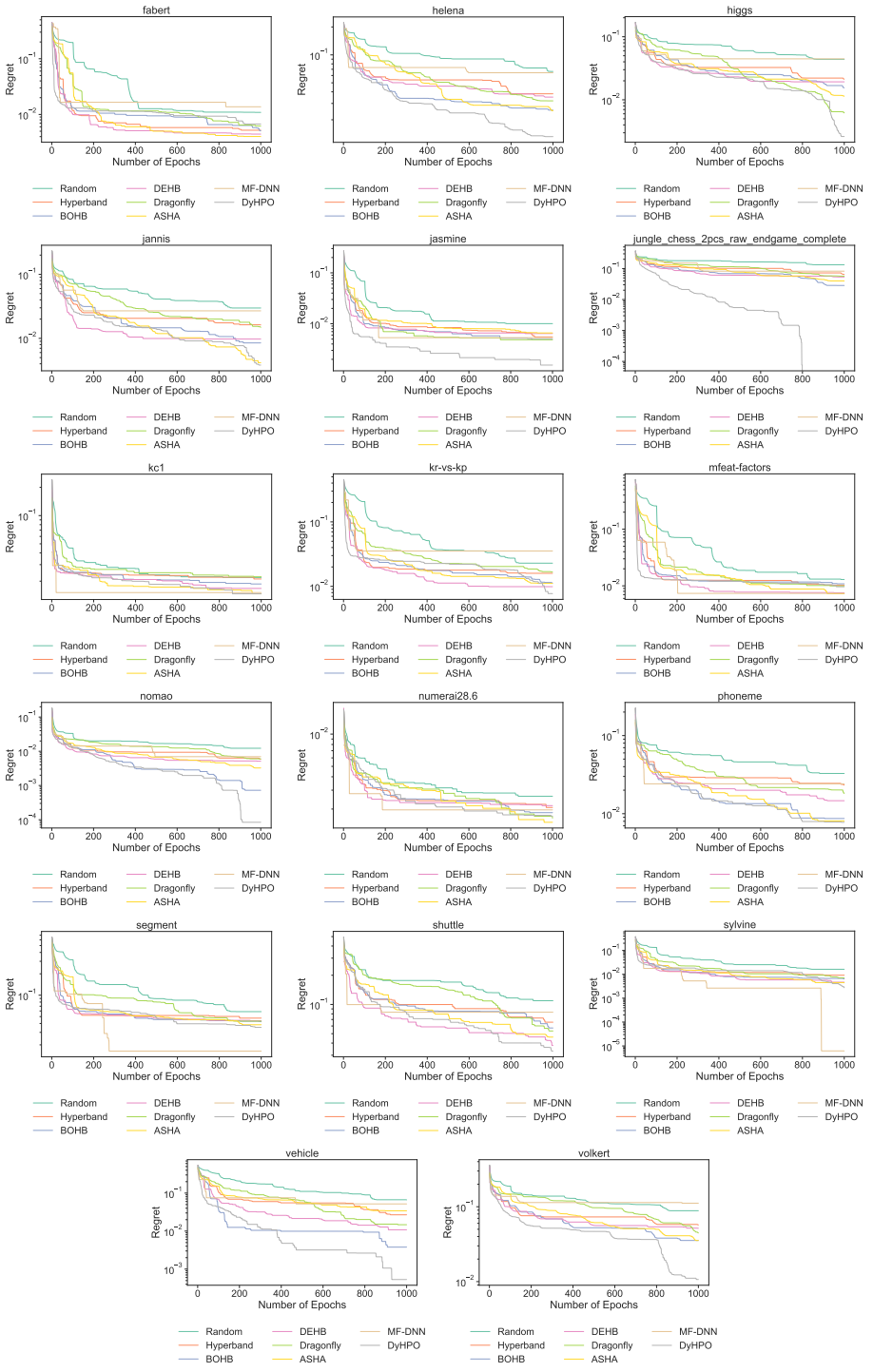


Figure 11: Performance comparison over the number of steps on a dataset level for LCBench (cont.).

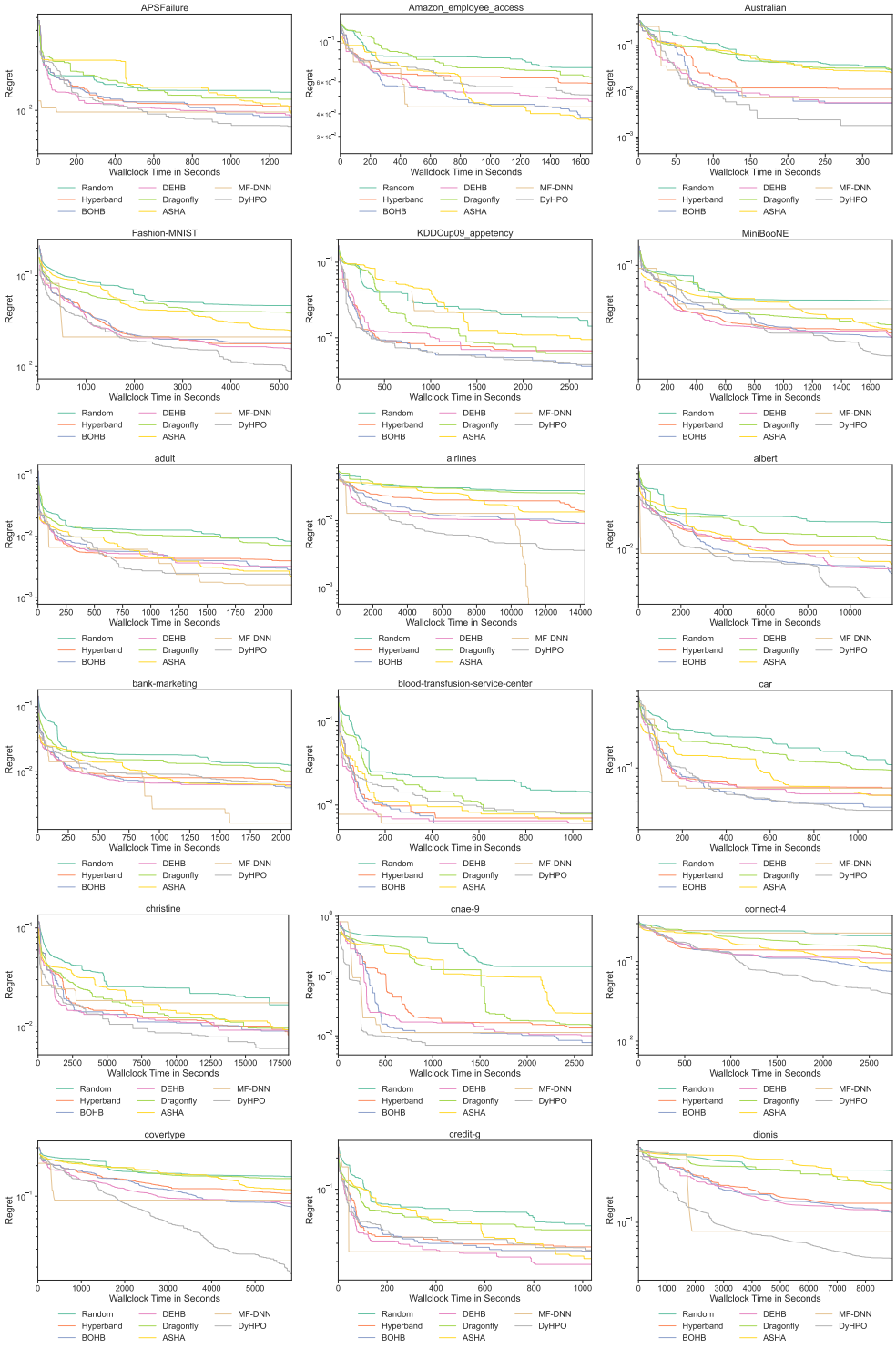


Figure 12: Performance comparison over time on a dataset level for LCBench with the overhead included.

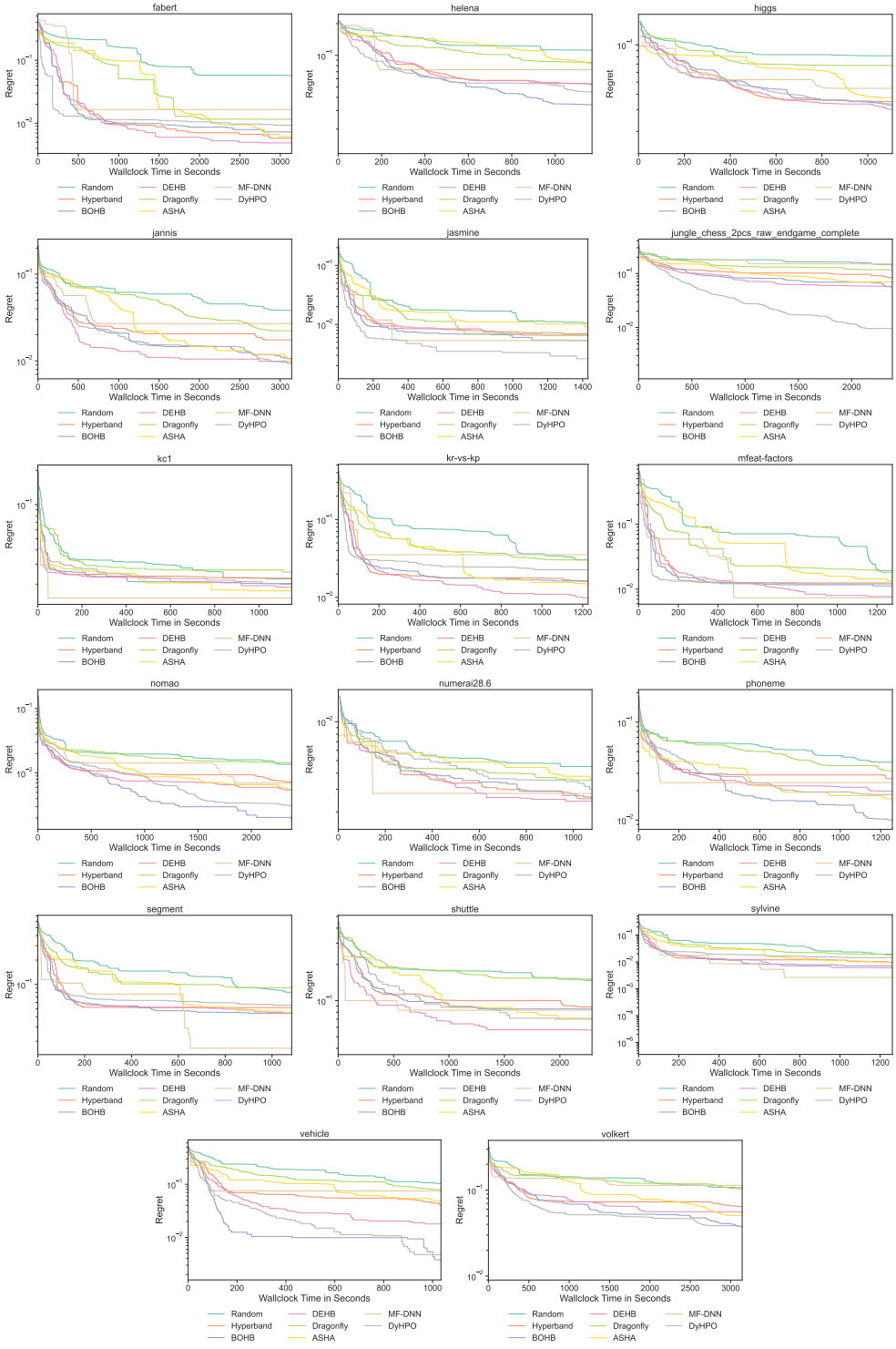


Figure 13: Performance comparison over time on a dataset level for LCBench with the overhead included. (cont.).