Cost-Sensitive Multi-Fidelity Bayesian Optimization with Transfer of Learning Curve Extrapolation

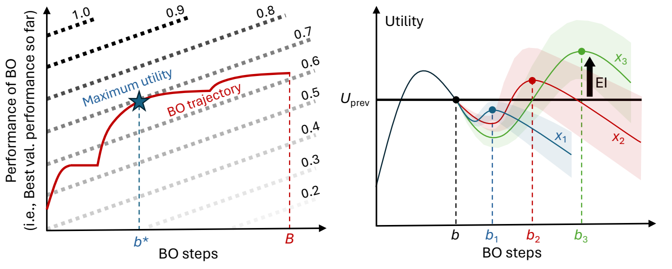
Dong Bok Lee[1] Aoxuan Silvia Zhang[2,*] Byungjoo Kim[1,*] Junhyeon Park[1,*] Juho Lee[1]
Sung Ju Hwang[1,3] Hae Beom Lee[1]

[*]Equal contribution.
[1]KAIST
[2]Korea University
[2]DeepAuto

Abstract In this paper, we address the problem of *cost-sensitive multi-fidelity* Bayesian Optimization (BO) for efficient hyperparameter optimization (HPO). Specifically, we assume a scenario where users want to early-stop the BO when the performance improvement is not satisfactory with respect to the required computational cost. To this end, we aim to explicitly improve such trade-off by dynamically finding the hyperparameter configurations that can maximally improve the trade-off in future, and also automatically stopping the BO around the best trade-off. Further, we improve the sample efficiency of learning curve (LC) extrapolation with transfer learning. We validate our method on various LC datasets, achieving significantly better trade-off than the baselines we consider.

1 Introduction

The goal of multi-fidelity (or gray-box) Bayesian optimization (BO) is to make use of lower fidelity information to predict and optimize the performances at higher or full fidelity, greatly improving the sample efficiency of BO (Li et al., 2018; Falkner et al., 2018; Awad et al., 2021; Swersky et al., 2014; Wistuba et al., 2022; Arango et al., 2023; Kadra et al., 2023; Rakotoarison et al., 2024). Unlike black-box BO, multi-fidelity BO dynamically selects hyperparameter configurations even before finishing a single training run, demonstrating its ability of finding better configurations in a more sample efficient manner than black-box BO. However, the existing methods are not aware of the trade-off between the cost and performance of BO – users may want to penalize the cost of BO with respect to its performance, and in this case the BO process should focus on exploiting the current belief on good hyperparameter configurations than trying to explore new configurations. Yet, existing multi-fidelity BO methods tend to over-explore because they usually assume a sufficiently large budget for the BO and aim to obtain the best asymptotic performance on a validation set, hence are not able to properly penalize the cost (Swersky et al., 2014).



(a) (b)

Figure 1: **(a)** A utility function shown in the dotted black lines. The red curve shows a BO trajectory from which we determine the maximum utility (≈ 0.7) and when to stop (b^*). **(b)** An illustration of selecting the best configuration at each BO step. Notice, the y-axis is utility. Starting from the current BO step b, we extrapolate the LCs with the three configurations x_1, x_2, x_3 (shown in the solid curves with colors and the shaded area), and then select x_3 which achieves at b_3 the maximum expected improvement (EI) of utility over the previous utility U_{prev}.

In this paper, we introduce a sophisticated notion of cost-sensitivity for multi-fidelity BO. Specifically, we introduce *utility*, a function which is predefined by each user and describes users' own preferences about the trade-off. It has higher values as cost decreases and performance increases, and vice versa (Fig. 1a). We explicitly maximize this utility by dynamically selecting hyperparamter configurations which are expected to maximally improve it in future (Fig. 1b), and also automatically terminating the BO around the maximum utility (Fig. 1a), instead of terminating at an arbitrary target budget. We call our method Cost-sensitive Multi-fidelity **BO** (CMBO). We first introduce the acquisition function and stopping criteria specifically developed for our framework, and explain how to achieve with them a good trade-off between cost and performance (utility) of multi-fidelity BO (§2.1). Also, based on a recently introduced Prior-Fitted Networks (PFNs) (Müller et al., 2021; Adriaensen et al., 2023) for in-context Bayesian inference, we explain how to transfer-learn a PFN with the existing learning curve (LC) datasets to develop a sample efficient in-context surrogate function for freeze-thaw BO that can also effectively capture the correlations between different hyperparameter configurations (§2.1). Lastly, we empirically demonstrate the superiority of CMBO on a set of diverse user preferences and three HPO benchmarks, showing that it significantly outperforms all the previous multi-fidelity BO and the transfer-BO baselines we consider (§3). We discuss related work in §A.

2 Approach

In this section, we introduce CMBO, a novel framework for cost-sensitive multi-fidelity BO.

Notation. Following the convention, we assume that we are given a finite pool of hyperparameter configurations $\mathcal{X} = \{x_1, \dots, x_N\}$, with N the number of configurations. Let $t \in [T] := \{1, \dots, T\}$ denote the training epochs, T the last epoch, and $y_{n,1}, \dots, y_{n,T}$ the validation performances (e.g., validation accuracies) obtained with the configuration x_n. We further introduce notations for multi-fidelity BO. Let $b = 1, \dots, B$ denote the BO steps, B the last BO step, and $\tilde{y}_1, \dots, \tilde{y}_B$ the BO performances, i.e., each $\tilde{y}_b$ is the best validation performance (y) obtained until the BO step b.

Freeze-thaw BO. Freeze-thaw BO (Swersky et al., 2014) is an advanced form of multi-fidelity BO. At each BO step, it allows us to dynamically select and evaluate the best hyperparameter configuration x_{n^*} with $n^* \in [N]$ denoting the corresponding index, while pausing the evaluation on the previous best configuration. Specifically, given $\mathcal{C} = \{(x, t, y)\}$ that represents a set of partial learning curves (LCs) collected up to a specific BO step, we predict for all $x \in \mathcal{X}$ the remaining part of the LCs up to the last training epoch T with a (pretrained) LC extrapolator, compute the acquisition such as the expected improvement (Mockus et al., 1978) of validation performance at epoch T, and select and evaluate the best configuration x_{n^*} that maximizes the acquisition. Note that at any BO step, the partial LCs in $\mathcal{C}$ can have different length across the configurations. Suppose that at BO step b the next training epoch for x_{n^*} is t_{n^*}. We then evaluate x_{n^*} a single epoch from the corresponding checkpoint to obtain the validation performance $y_{n^*,t_{n^*}}$ at the next epoch t_{n^*}, which we use to update the corresponding partial LC in $\mathcal{C}$ and compute the BO performance $\tilde{y}_b$. We repeat this process B times until convergence. See Alg. 1 for the pseudocode (except the red parts).

2.1 Cost-sensitive Multi-fidelity BO

We next introduce our main method and algorithm based on freeze-thaw BO.

Utility function.. A utility function U[1] describes the trade-off between the BO step b and the BO performance $\tilde{y}_b$. Its values $U(b, \tilde{y}_b)$ negatively correlate with b and positively with $\tilde{y}_b$. For instance, we may simply define $U(b, \tilde{y}_b) = \tilde{y}_b - \alpha b$ for some $\alpha > 0$, such that the utility gives linear incentives and penalties to the performance and number of BO steps, respectively.

[1]In this paper we assume that the utility function is predefined by a user. One can instead try to learn it from data, but we leave that as a future work.

Acquisition function.. Let t_n be the next training epoch for the configuration x_n at a BO step b. Further, suppose we have a LC extrapolator $f(\cdot|x_n, \mathcal{C})$ that can probabilistically estimate x_n's remaining part of LC, $y_{n,t_n:T}$, conditioned on $\mathcal{C}$ a set of partial LCs collected up to the step b. Then, based on the expected improvement (EI) (Mockus et al., 1978), we define the acquisition function A:

$$A(n) = \max_{\Delta t \in \{0,\ldots,T-t_n\}} \mathbb{E}_{y_{n,t_n:T} \sim f(\cdot|x_n,\mathcal{C})} \left[\max\left(0, U(b+\Delta t, \tilde{y}_{b+\Delta t}) - U_{\text{prev}}\right) \right]. \tag{1}$$

In Eq. (1), we first extrapolate $y_{n,t_n:T}$, the remaining part of the LC associated with x_n, and compute the corresponding predictive BO performances $\{\tilde{y}_{b+\Delta t} \mid \Delta t = 0, \ldots, T - t_n\}$, where $\tilde{y}_{b+\Delta t}$ is computed by taking the maximum among the last step BO performance $\tilde{y}_{b-1}$ as well as the newly extrapolated validation performances $y_{n,t_n}, \ldots, y_{n,t_n+\Delta t}$. Then, based on the increased BO step $b + \Delta t$ and the updated BO performance $\tilde{y}_{b+\Delta t}$, we compute the corresponding utility, and its expected improvement over the previous utility U_{prev} over the distribution of LC extrapolation with the Monte-Carlo (MC) estimation. The acquisition $A(n)$ is defined by picking the best increment $\Delta t \in \{0, \ldots, T - t_n\}$ that maximizes the expected improvement, and we eventually choose the best configuration index n that maximizes A (see Fig. 1b).

Algorithm 1 Cost-sensitive Multi-fidelity BO

```
1:  Input: LC extrapolator $f$, acquisition function $A$, utility function $U$, maximum BO steps $B$, hyperparameter
    configuration pool $\mathcal{X}$, number of configurations $N$.
2:  $U_{\text{prev}} \leftarrow 0$, $\tilde{y}_0 \leftarrow -\infty$, $\mathcal{C} \leftarrow \emptyset$, $t_1, \ldots, t_N \leftarrow 1$
3:  for $b = 1, \ldots, B$ do
4:      $n^* \leftarrow \arg\max_n A(n)$                ▷ Acquisition func., Eq. (1)
5:      if Eq. (2) and $b > 1$ then                     ▷ Stopping criterion
6:          Break the for loop                          ▷ Stop the BO
7:      end if
8:      Evaluate $y_{n^*,t_{n^*}}$ with $x_{n^*}$.
9:      $\mathcal{C} \leftarrow \mathcal{C} \cup \{(x_{n^*}, t_{n^*}, y_{n^*,t_{n^*}})\}$    ▷ Update the history
10:     $\tilde{y}_b \leftarrow \max(\tilde{y}_{b-1}, y_{n^*,t_{n^*}})$          ▷ Update the BO perf.
11:     $U_{\text{prev}} \leftarrow U(b, \tilde{y}_b)$                  ▷ Update the prev. utility
12:     $t_{n^*} \leftarrow t_{n^*} + 1$
13: end for
```

Stopping criterion. The next question is how to properly stop the BO around the maximum utility. We propose to stop when the following criterion is satisfied at each BO step $b > 1$:

$$(\hat{U}_{\max} - U_{\text{prev}}) / (\hat{U}_{\max} - \hat{U}_{\min}) > \delta_b. \tag{2}$$

In Eq. (2), U_{prev} is the utility value at the last step $b-1$, $\hat{U}_{\max}$ is defined as the maximum utility value seen up to the last step, and $\hat{U}_{\min} = U(B, \tilde{y}_1)$. The role of $\hat{U}_{\max}$ and $\hat{U}_{\min}$ is to roughly estimate the maximum and the minimum utility achievable over the course of BO, respectively. Therefore, the LHS of Eq. (2) can be seen as the *normalized regret* of utility roughly estimated at the current step b, and we stop the BO as soon as the current estimation on the regret exceeds some threshold δ_b.

To define δ_b, let $n^* = \arg\max_n A(n)$ denote the index of the currently chosen best configuration x_{n^*} based on Eq. (1), BetaCDF is the CDF of Beta, and $\mathbb{1}$ the indicator function. Then, we have:

$$\delta_b = \text{BetaCDF}(p_b; \beta, \beta)^{\gamma}, \quad \beta, \gamma > 0, \tag{3}$$

$$p_b = \max_{\Delta t \in \{0,\ldots,T-t_{n^*}\}} \mathbb{E}_{y_{n,t_{n^*}:T} \sim f(\cdot|x_{n^*},\mathcal{C})} \left[\mathbb{1}\left(U\left(b+\Delta t, \tilde{y}_{b+\Delta t}\right) > U_{\text{prev}}\right) \right]. \tag{4}$$

p_b in Eq. (4) is the probability that the current best configuration x_{n^*} improves on U_{prev} in some future BO step (i.e., probability of improvement, or PI (Mockus et al., 1978)). Intuitively, we want to defer the termination as p_b increases, and vice versa. It is considered in Eq. (3) – as p_b increases, the threshold δ_b increases as well because $\text{BetaCDF}(\cdot; \beta, \beta)^{\gamma}$ is a monotonically increasing function in $[0, 1]$, so we have less motivation to stop according to Eq. (2). Therefore, our stopping criterion can be seen as a smooth interpolation between two extreme stopping criteria: 1. normalized regret and 2. probability of improvement, with β and γ determining the shape of the interpolation. We discuss the effects of β and γ on our stopping criterion in §B.

Table 1: **Results on the cost-sensitive multi-fidelity HPO setups** ($\alpha = 4e\text{-}05, 2e\text{-}04$). We multiply 100 to the normalized regret. The transfer learning methods are indicated by blue.

Method	LCBench				TaskSet				PD1			
	$\alpha = 4e\text{-}05$		$\alpha = 2e\text{-}04$		$\alpha = 4e\text{-}05$		$\alpha = 2e\text{-}04$		$\alpha = 4e\text{-}05$		$\alpha = 2e\text{-}04$	
	Regret	Rank	Regret	Rank	Regret	Rank	Regret	Rank	Regret	Rank	Regret	Rank
Random (Bergstra and Bengio, 2012)	$14.1_{\pm 1.8}$	7.7	$18.1_{\pm 1.7}$	7.5	$18.9_{\pm 4.6}$	7.8	$22.3_{\pm 4.3}$	7.7	$5.4_{\pm 2.3}$	7.1	$11.0_{\pm 5.6}$	7.2
ASHA (Li et al., 2020a)	$8.6_{\pm 1.0}$	6.3	$13.7_{\pm 1.1}$	6.8	$8.4_{\pm 4.1}$	6.2	$14.7_{\pm 5.4}$	7.0	$5.9_{\pm 7.3}$	6.5	$9.7_{\pm 7.4}$	6.5
BOHB (Falkner et al., 2018)	$6.4_{\pm 1.0}$	5.0	$11.6_{\pm 1.0}$	5.5	$7.4_{\pm 1.4}$	6.9	$11.2_{\pm 1.9}$	6.2	$1.6_{\pm 0.2}$	4.7	$4.9_{\pm 0.2}$	4.9
DEHB (Awad et al., 2021)	$6.1_{\pm 1.6}$	4.6	$11.0_{\pm 1.4}$	4.9	$5.8_{\pm 2.3}$	6.1	$10.0_{\pm 1.7}$	6.0	$2.1_{\pm 0.1}$	6.1	$5.4_{\pm 0.1}$	6.0
DyHPO (Wistuba et al., 2022)	$7.2_{\pm 1.2}$	5.7	$12.1_{\pm 1.6}$	5.9	$7.5_{\pm 2.1}$	6.4	$11.1_{\pm 2.0}$	6.3	$2.5_{\pm 0.6}$	6.2	$6.2_{\pm 0.9}$	6.7
DPL (Kadra et al., 2023)	$3.8_{\pm 0.5}$	3.2	$9.3_{\pm 0.5}$	4.4	$2.6_{\pm 0.7}$	3.4	$7.5_{\pm 0.6}$	4.5	$1.8_{\pm 0.3}$	4.5	$5.1_{\pm 0.6}$	4.7
Quick-Tune† (Arango et al., 2023)	$9.6_{\pm 0.0}$	6.9	$12.7_{\pm 0.0}$	6.1	$3.7_{\pm 0.0}$	3.7	$5.6_{\pm 0.0}$	3.1	$2.4_{\pm 0.0}$	5.4	$5.5_{\pm 0.0}$	5.0
FSBO (Wistuba and Grabocka, 2020)	$2.6_{\pm 0.0}$	3.0	$6.4_{\pm 0.0}$	2.7	$2.9_{\pm 0.0}$	3.0	$4.9_{\pm 0.0}$	2.6	$1.3_{\pm 0.0}$	2.6	$4.2_{\pm 0.0}$	3.1
CMBO (ours)	$\mathbf{2.3}_{\pm 0.1}$	**2.7**	$\mathbf{3.1}_{\pm 0.0}$	**1.3**	$\mathbf{1.3}_{\pm 0.0}$	**1.5**	$\mathbf{3.1}_{\pm 1.0}$	**1.5**	$\mathbf{0.8}_{\pm 0.0}$	**1.9**	$\mathbf{0.9}_{\pm 0.0}$	**1.0**

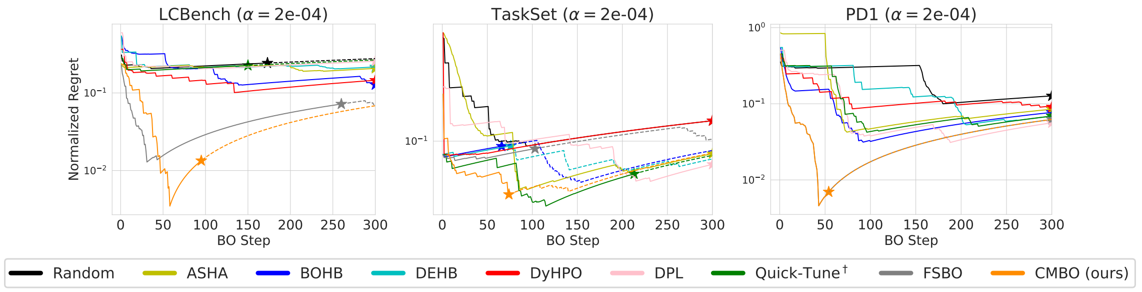


Figure 2: **Visualization of the normalized regret over BO steps** on $\alpha = 2e\text{-}04$. The asterisks indicate the stopping points, and the dotted lines represent the normalized regret achievable by running each method without stopping. See §H for the results on all the other tasks.

Algorithm. See Alg. 1 for the pseudocode, with the red parts corresponding the specifics of ours.

Transfer learning. Since users may want to early-stop the BO, we should have a sample efficient LC extrapolation mechanism for preventing inaccurate early-stopping before collecting a sufficient amount of BO observations. We thus propose to make use of *transfer learning* with mixup (Zhang et al., 2018) to improve the sample efficiency of the LC extrapolator. We defer more details on our mixup strategy and training of the LC extrapolator to §D and §E, respectively.

3 Experiments

Datasets and baselines. We use three benchmark datasets including **LCBench** (Zimmer et al., 2021), **TaskSet** (Metz et al., 2020), and **PD1** (Wang et al., 2021). For the baselines. we compare our method against **Random Search** (Bergstra and Bengio, 2012), **ASHA** (Li et al., 2020a), **BOHB** (Falkner et al., 2018), **DEHB** (Awad et al., 2021), **DyHPO** (Wistuba et al., 2022), **DPL** (Kadra et al., 2023), **Quick-Tune**† (Arango et al., 2023), and **FSBO** (Wistuba and Grabocka, 2020). See §C and §F for more details on datasets and baselines, respectively.

Experimental setups. We use a linear function for penalizing the cost of multi-fidelity BO, i.e., $U(b, \tilde{y}) = \tilde{y} - \alpha b$ where $\tilde{y}$ is the BO performance, b the BO steps, and $\alpha \in \{0, 4e\text{-}05, 2e\text{-}04\}$. For the baselines, we simply use the fixed threshold $\delta_b = 0.2$ in Eq. (2) as computing the PI in Eq. (4) is not straightforward for them. For our model, we use $\beta = \exp(3)$ and $\gamma = \log_2 5$ for all the experiments in this paper, except the ablation study in Fig. 3d. In order to report the average performances over the tasks, we use the normalized regret of utility $(U_{\max} - U_{b^*})/(U_{\max} - U_{\min}) \in [0, 1]$, similarly to Eq. (2). We also report the rank of each method averaged over the tasks. Please see §G for more details on experimental setups such as how to compute $U_{\max}$ and $U_{\min}$.

Results. Table 1 shows the performance of each method on the cost-sensitive multi-fidelity HPO setup ($\alpha > 0$). We see that our method largely outperforms all the methods on all the settings, including the multi-fidelity HPO and the transfer-BO methods, in terms of both normalized regret and average rank. Notice, our method achieves better average rank as the penalty becomes stronger

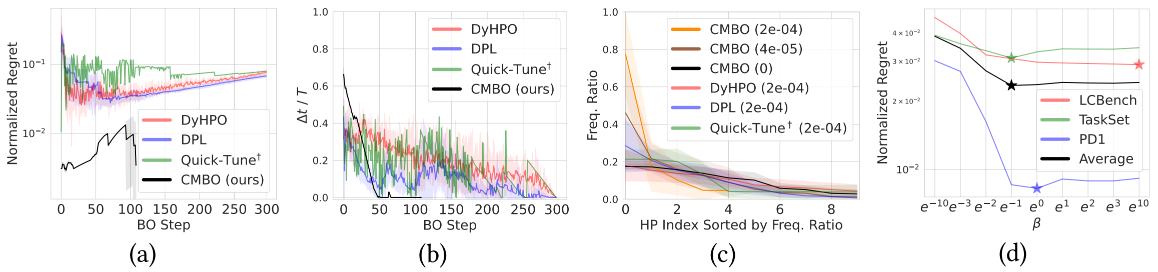


Figure 3: **(a, b, c)** Additional analysis on the effectiveness of our acquisition function. We use PD1 for the visualization. In **(c)**, the values of α are shown in the parenthesis. **(d)** Ablation study on β, with the minimum regret shown with the asterisks.

(α = 2e-04). Fig. 2 illustrates the normalized regret over the course of BO, where our method achieves significantly lower regret upon termination. Our method tends to achieve the minimum regret earlier than the baselines, demonstrating its sample efficiency in searching good hyperparameter configurations by explicitly considering the utility during the BO. In §H, we provide additional experimental results including various ablation studies that support the efficacy of our method.

Analysis. In order to clearly understand the source of improvements, we next analyze the configurations chosen by each method. Specifically, for each BO step b, we run the configuration currently selected at step b up to its last epoch T, and compute its minimum ground-truth regret achievable at some future step $b + \Delta t$ (Fig. 3a), as well as the corresponding optimal increment Δt (Fig. 3b). In Fig. 3a, our method shows much lower minimum regret than the baselines. It means that our acquisition function in Eq. (1) works as intended, trying to select at each BO step the best configuration which is expected to maximally improve the utility in future. Fig. 3b shows that the configurations chosen by our method initially correspond to greater Δt (i.e., non-greedy), but gradually to the smaller Δt (i.e., greedy). It is because as the BO proceeds, the performance improvements of BO saturate, so the cost of BO quickly dominates the trade-off, leading to smaller Δt even close to 0. On the other hand, the tendencies of the baselines are very noisy and relatively unclear. Lastly, Fig. 3c shows the distribution of the top-10 most frequently selected configurations during the BO. As expected, our method tend to focus only on a few configurations during the BO to maximize the short-term performances, especially when the penalty is stronger with greater α. On the other hand, the baselines tend to overly explore the configurations even when the penalty is the strongest (α = 2e-04).

Lastly, we analyze the effectiveness of our stopping criterion discussed in Eq. (2), (3), and (4). Fig. 3d shows the normalized regret over the different values of β, a mixing coefficient between the two extreme stopping criteria, as discussed in §2.1. $\beta \longrightarrow 0$ corresponds to the criterion used by the baselines which is only based on the estimated normalized regret, whereas $\beta \longrightarrow +\infty$ corresponds to the hard thresholding only based on the PI. We can see that the optimal criterion is achieved by smoothly mixing between the two ($\beta = e^{-1}$), demonstrating the superiority of our stopping criterion to the one used by the baselines ($\beta \longrightarrow 0$).

4 Broader Impact and Limitation

In this paper, we discussed cost-sensitive multi-fidelity BO (CMBO), a novel framework for improving the sample efficiency of HPO. The proposed CMBO is designed to maximize user utility which defines tradeoff between computational cost and performance and to stop searching hyperparameter; therefore, we can save certain amounts of computational resources (e.g., runtime of GPUs). This reduces carbon emissions which is closely related to the most arising issue of global warming.

Although our method sheds light on improving the efficiency of HPO, there remain a few limitations. First, we assumed that the utility function is given, but instead we could learn it from data provided by each user. Second, our LC extrapolator is prone to overfitting even with the mixup strategy when the training dataset is small. Thus, we need a theoretically grounded way to incorporate the prior on LCs and infer the corresponding posterior.

References

Abdolshah, M., Shilton, A., Rana, S., Gupta, S., and Venkatesh, S. (2019). Cost-aware multi-objective bayesian optimisation. *arXiv preprint arXiv:1909.03600.*

Adriaensen, S., Rakotoarison, H., Müller, S., and Hutter, F. (2023). Efficient bayesian learning curve extrapolation using prior-data fitted networks. *Advances in Neural Information Processing Systems*, 36.

Arango, S. P., Ferreira, F., Kadra, A., Hutter, F., and Grabocka, J. (2023). Quick-tune: Quickly learning which pretrained model to finetune and how. In *The Twelfth International Conference on Learning Representations.*

Awad, N., Mallik, N., and Hutter, F. (2021). Dehb: Evolutionary hyberband for scalable, robust and efficient hyperparameter optimization. In Zhou, Z.-H., editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 2147–2153. International Joint Conferences on Artificial Intelligence Organization. Main Track.

Bai, T., Li, Y., Shen, Y., Zhang, X., Zhang, W., and Cui, B. (2023). Transfer learning for bayesian optimization: A survey. *arXiv preprint arXiv:2302.05927.*

Baker, B., Gupta, O., Raskar, R., and Naik, N. (2017). Accelerating neural architecture search using performance prediction. *arXiv preprint arXiv:1705.10823.*

Bergstra, J., Bardenet, R., Bengio, Y., and Kégl, B. (2011). Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, 24.

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2).

Cowen-Rivers, A. I., Lyu, W., Tutunov, R., Wang, Z., Grosnit, A., Griffiths, R. R., Maraval, A. M., Jianye, H., Wang, J., Peters, J., et al. (2022). Hebo: Pushing the limits of sample-efficient hyper-parameter optimisation. *Journal of Artificial Intelligence Research*, 74:1269–1349.

Domhan, T., Springenberg, J. T., and Hutter, F. (2015). Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Twenty-fourth international joint conference on artificial intelligence.*

Falkner, S., Klein, A., and Hutter, F. (2018). Bohb: Robust and efficient hyperparameter optimization at scale. In *International conference on machine learning*, pages 1437–1446. PMLR.

Gargiani, M., Klein, A., Falkner, S., and Hutter, F. (2019). Probabilistic rollouts for learning curve extrapolation across hyperparameter settings. *arXiv preprint arXiv:1910.04522.*

Garnelo, M., Schwarz, J., Rosenbaum, D., Viola, F., Rezende, D. J., Eslami, S., and Teh, Y. W. (2018). Neural processes. *arXiv preprint arXiv:1807.01622.*

Golovin, D., Solnik, B., Moitra, S., Kochanski, G., Karro, J., and Sculley, D. (2017). Google vizier: A service for black-box optimization. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1487–1495.

Hendrycks, D. and Gimpel, K. (2016). Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415.*

Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration. In *Learning and Intelligent Optimization: 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers 5*, pages 507–523. Springer.

Hvarfner, C., Stoll, D., Souza, A., Lindauer, M., Hutter, F., and Nardi, L. (2021). $\backslash pi$ bo: Augmenting acquisition functions with user beliefs for bayesian optimization. In *International Conference on Learning Representations.*

Kadra, A., Janowski, M., Wistuba, M., and Grabocka, J. (2023). Scaling laws for hyperparameter optimization. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Karnin, Z., Koren, T., and Somekh, O. (2013). Almost optimal exploration in multi-armed bandits. In *International conference on machine learning*, pages 1238–1246. PMLR.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Klein, A., Falkner, S., Bartels, S., Hennig, P., and Hutter, F. (2017a). Fast bayesian optimization of machine learning hyperparameters on large datasets. In *Artificial intelligence and statistics*, pages 528–536. PMLR.

Klein, A., Falkner, S., Springenberg, J. T., and Hutter, F. (2017b). Learning curve prediction with bayesian neural networks. In *International conference on learning representations*.

Krizhevsky, A., Hinton, G., et al. (2009). Learning multiple layers of features from tiny images.

Lee, E. H., Perrone, V., Archambeau, C., and Seeger, M. (2020). Cost-aware bayesian optimization. *arXiv preprint arXiv:2003.10870*.

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. (2018). Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18(185):1–52.

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, S., Xing, W., Kirby, R., and Zhe, S. (2020b). Multi-fidelity bayesian optimization via deep neural networks. *Advances in Neural Information Processing Systems*, 33:8521–8531.

Mallik, N., Bergman, E., Hvarfner, C., Stoll, D., Janowski, M., Lindauer, M., Nardi, L., and Hutter, F. (2024). Priorband: Practical hyperparameter optimization in the age of deep learning. *Advances in Neural Information Processing Systems*, 36.

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. *arXiv preprint arXiv:2002.11887*.

Mockus, J., Tiesis, V., and Zilinskas, A. (1978). The application of bayesian methods for seeking the extremum, vol. 2. *L Dixon and G Szego. Toward Global Optimization*, 2.

Müller, S., Feurer, M., Hollmann, N., and Hutter, F. (2023). Pfns4bo: In-context learning for bayesian optimization. In *International Conference on Machine Learning*, pages 25444–25470. PMLR.

Müller, S., Hollmann, N., Arango, S. P., Grabocka, J., and Hutter, F. (2021). Transformers can do bayesian inference. In *International Conference on Learning Representations*.

Nguyen, T. and Grover, A. (2022). Transformer neural processes: Uncertainty-aware meta learning via sequence modeling. In *International Conference on Machine Learning*, pages 16569–16594. PMLR.

Perrone, V., Jenatton, R., Seeger, M. W., and Archambeau, C. (2018). Scalable hyperparameter transfer learning. *Advances in neural information processing systems*, 31.

Poloczek, M., Wang, J., and Frazier, P. (2017). Multi-information source optimization. *Advances in neural information processing systems*, 30.

Rakotoarison, H., Adriaensen, S., Mallik, N., Garibov, S., Bergman, E., and Hutter, F. (2024). In-context freeze-thaw bayesian optimization for hyperparameter optimization. *arXiv preprint arXiv:2404.16795*.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252.

Salinas, D., Seeger, M., Klein, A., Perrone, V., Wistuba, M., and Archambeau, C. (2022). Syne tune: A library for large scale hyperparameter tuning and reproducible research. In *International Conference on Automated Machine Learning, AutoML 2022*.

Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25.

Snoek, J., Rippel, O., Swersky, K., Kiros, R., Satish, N., Sundaram, N., Patwary, M., Prabhat, M., and Adams, R. (2015). Scalable bayesian optimization using deep neural networks. In *International conference on machine learning*, pages 2171–2180. PMLR.

Snoek, J., Swersky, K., Zemel, R., and Adams, R. (2014). Input warping for bayesian optimization of non-stationary functions. In *International conference on machine learning*, pages 1674–1682. PMLR.

Souza, A., Nardi, L., Oliveira, L. B., Olukotun, K., Lindauer, M., and Hutter, F. (2021). Bayesian optimization with a prior for the optimum. In *Machine Learning and Knowledge Discovery in Databases. Research Track: European Conference, ECML PKDD 2021, Bilbao, Spain, September 13–17, 2021, Proceedings, Part III 21*, pages 265–296. Springer.

Swersky, K., Snoek, J., and Adams, R. P. (2013). Multi-task bayesian optimization. *Advances in neural information processing systems*, 26.

Swersky, K., Snoek, J., and Adams, R. P. (2014). Freeze-thaw bayesian optimization. *arXiv preprint arXiv:1406.3896*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

Wang, Z., Dahl, G. E., Swersky, K., Lee, C., Nado, Z., Gilmer, J., Snoek, J., and Ghahramani, Z. (2021). Pre-trained gaussian processes for bayesian optimization. *arXiv preprint arXiv:2109.08215*.

Wei, Y., Zhao, P., and Huang, J. (2021). Meta-learning hyperparameter performance prediction with neural processes. In *International Conference on Machine Learning*, pages 11058–11067. PMLR.

Wilson, A. G., Hu, Z., Salakhutdinov, R., and Xing, E. P. (2016). Deep kernel learning. In *Artificial intelligence and statistics*, pages 370–378. PMLR.

Wistuba, M. and Grabocka, J. (2020). Few-shot bayesian optimization with deep kernel surrogates. In *International Conference on Learning Representations*.

Wistuba, M., Kadra, A., and Grabocka, J. (2022). Supervising the multi-fidelity race of hyperparameter configurations. *Advances in Neural Information Processing Systems*, 35:13470–13484.

Wistuba, M. and Pedapati, T. (2020). Learning to rank learning curves. In *International Conference on Machine Learning*, pages 10303–10312. PMLR.

Zhang, H., Cisse, M., Dauphin, Y. N., and Lopez-Paz, D. (2018). mixup: Beyond empirical risk minimization. In *International Conference on Learning Representations*.

Zimmer, L., Lindauer, M., and Hutter, F. (2021). Auto-pytorch: Multi-fidelity metalearning for efficient and robust autodl. *IEEE transactions on pattern analysis and machine intelligence*, 43(9):3079–3090.

Submission 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] We provide several empirical justifications in §3 and §H.

   (b) Did you describe the limitations of your work? [Yes] See §4.

   (c) Did you discuss any potential negative societal impacts of your work? [N/A] The topic of our paper is about improving the efficiency of hyperparameter optimization, which is irrelevant to societal impacts.

   (d) Did you read the ethics review guidelines and ensure that your paper conforms to them? `https://2022.automl.cc/ethics-accessibility/` [Yes]

2. If you ran experiments...

   (a) Did you use the same evaluation protocol for all methods being compared (e.g., same benchmarks, data (sub)sets, available resources)? [Yes] We provide the details in §C and §G.

   (b) Did you specify all the necessary details of your evaluation (e.g., data splits, pre-processing, search spaces, hyperparameter tuning)? [Yes] We provide the details in §C and §G.

   (c) Did you repeat your experiments (e.g., across multiple random seeds or splits) to account for the impact of randomness in your methods or data? [Yes] We repeat our experiments for five times.

   (d) Did you report the uncertainty of your results (e.g., the variance across random seeds or splits)? [Yes] We provide standard deviation for five runs.

   (e) Did you report the statistical significance of your results? [Yes] We provide standard deviation for five runs.

   (f) Did you use tabular or surrogate benchmarks for in-depth evaluations? [Yes] We use three HPO public benchmarks.

   (g) Did you compare performance over time and describe how you selected the maximum duration? [Yes] We provide the utility over time in §H.

   (h) 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] We provide the details in §G.

   (i) Did you run ablation studies to assess the impact of different components of your approach? [Yes] We provide the results of ablation studies in §H.

3. With respect to the code used to obtain your results...

   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results, including all requirements (e.g., `requirements.txt` with explicit versions), random seeds, an instructive `README` with installation, and execution commands (either in the supplemental material or as a URL)? [N/A] We provide anonymized research code in supplemental materials. We are going to release our code upon the acceptance.

   (b) Did you include a minimal example to replicate results on a small subset of the experiments or on toy data? [Yes] We provide an example in the code of supplemental materials.

   (c) Did you ensure sufficient code quality and documentation so that someone else can execute and understand your code? [N/A] We are going to release code and usage documentation through github upon the acceptance.

   (d) Did you include the raw results of running your experiments with the given code, data, and instructions? [Yes] We provide the full utility trajectory without cherry-picking in §H.

   (e) Did you include the code, additional data, and instructions needed to generate the figures and tables in your paper based on the raw results? [N/A] We are going to release them through github upon the acceptance.

4. If you used existing assets (e.g., code, data, models)…

   (a) Did you cite the creators of used assets? [Yes] We all cite the creators in §3.

   (b) Did you discuss whether and how consent was obtained from people whose data you're using/curating if the license requires it? [N/A] Those benchmarks we use in this paper are fully public.

   (c) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] Those benchmarks we use in this paper are learning curve datasets.

5. If you created/released new assets (e.g., code, data, models)…

   (a) Did you mention the license of the new assets (e.g., as part of your code submission)? [N/A] We are going to mention the license upon the acceptance through github.

   (b) Did you include the new assets either in the supplemental material or as a URL (to, e.g., GitHub or Hugging Face)? [N/A] We are going to include all the new assets upon the acceptance through github.

6. 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] We did not use crowdsourcing or conducted research with human subjects.

   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A] We did not use crowdsourcing or conducted research with human subjects.

   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A] We did not use crowdsourcing or conducted research with human subjects.

7. If you included theoretical results…

   (a) Did you state the full set of assumptions of all theoretical results? [N/A] We did not include theoretical results.

   (b) Did you include complete proofs of all theoretical results? [N/A] We did not include theoretical results.

A Related Work

Multi-fidelity HPO. Unlike traditional black-box approaches for HPO (Bergstra and Bengio, 2012; Hutter et al., 2011; Bergstra et al., 2011; Snoek et al., 2012, 2015, 2014; Cowen-Rivers et al., 2022; Müller et al., 2023), multi-fidelity (or gray-box) HPO aims to optimize hyperparameters in a sample efficient manner by utilizing low fidelity information (e.g., validation set performances with smaller training dataset) as a proxy for higher or full fidelities (Swersky et al., 2013; Klein et al., 2017a; Poloczek et al., 2017; Li et al., 2020b), dramatically speeding up the HPO. In this paper, we focus on making use of performances at fewer training epochs to better predict/optimize the performances at longer training epochs. One of the well-known examples is Hyperband (Li et al., 2018), a bandit-based method that randomly selects a set of random hyperparameter configurations, and stops poorly performing ones using successive halving (Karnin et al., 2013) even before reaching the last training epoch. While Hyperband shows much better performance than random search (Bergstra and Bengio, 2012), its computational or sample efficiency can be further improved by replacing random sampling of configurations with Bayesian optimization (Falkner et al., 2018), adopting evolution strategy to promote internal knowledge transfer (Awad et al., 2021), or making it asynchronously parallel (Li et al., 2020a).

Freeze-thaw BO. Whereas the form of knowledge transfer of Hyperband (and its variants) from lower to higer fidelity is indirect, freeze-thaw BO (Swersky et al., 2014) transfers knowledge more directly by explicitly modeling a GP kernel to jointly model interactions between different training budgets and configurations. It then dynamically pauses (freezes) and resumes (thaws) configurations based on the last epoch performances extrapolated from a set of partially observed LCs obtained from other configurations, leading to an efficient and sensible allocation of computational resources. DyHPO (Wistuba et al., 2022) and its transfer version (Arango et al., 2023) improve the computational efficiency of freeze-thaw BO (Swersky et al., 2014) with deep kernel GP (Wilson et al., 2016), but their multi-fidelity version of expected improvement (EI) acquisition extrapolates the LCs only a one-step forward, producing a greedy strategy for dynamically selecting configurations. Other recent variants of freeze-thaw BO include DPL (Kadra et al., 2023) and ifBO (Rakotoarison et al., 2024) which are not greedy and show competitive performances, but they are either lack of transfer learning (Kadra et al., 2023; Rakotoarison et al., 2024), resort to too strong assumptions on the shape of LCs (Rakotoarison et al., 2024), or incur the cost of retraining the surrogate function for each BO step (Rakotoarison et al., 2024). On the other hand, we maximize the sample efficiency of freeze-thaw BO with transfer learning, while getting rid of any need for such strong assumptions or retraining costs.

Learning curve extrapolation. Freeze-thaw BO requires the ability of dynamically updating predictions on future performances from a growing set of partially observed LCs, thus heavily relies on the ability of LC extrapolation (Baker et al., 2017; Gargiani et al., 2019; Wistuba and Pedapati, 2020). The LC extrapolation used in DyHPO (Wistuba et al., 2022) and Quick-Tune (Arango et al., 2023) is based on deep kernel GP (Wilson et al., 2016), but extrapolates only a single step forward, making it hard to be used for our case - maximizing utilites at any possible future time steps. Freeze-thaw BO (Swersky et al., 2014) and DPL (Kadra et al., 2023) use non-greedy extrapolations but limit the shape of LCs with exponential decay kernel or power law functions, which is questionable if such strong inductive biases are applicable to more general cases. Domhan et al. (2015) consider a broader set of basis functions to define a prior on LCs and infer its posterior, but requires computationally expensive MCMC, and also do not consider correlations between different configurations. Klein et al. (2017b) models interactions between configurations with a Bayesian neural network (BNN), but suffers from the same computational inefficiency of MCMC and also the additional cost for online retraining of the BNN. LC-PFNs (Adriaensen et al., 2023) are an in-context Bayesian LC extrapolation method without retraining, based on recently introduced Prior-Fitted Networks (PFNs) (Müller et al., 2021). However, as with Domhan et al. (2015), LC-PFNs do not consider interactions between configurations and hence is suboptimal to use as a surrogate function for freeze-thaw BO. Recently, Rakotoarison et al. (2024) further combine LC-PFN with PFN4BO (Müller et al., 2023) to develop an in-context surrogate function for freeze-thaw BO, but they train PFNs only with a prior distribution similarly to the original PFN training scheme. On the other hand, we use transfer learning, i.e., train PFNs with the existing LC datasets, to improve the sample efficiency of freeze-thaw BO while successfully encoding the correlations between configurations at the same time.

Transfer BO. Transfer learning can be used for improving the sample efficiency of BO (Bai et al., 2023), and here we list a few of them. Some of recent works explore scalable transfer learning with deep neural networks (Perrone et al., 2018; Wistuba and Grabocka, 2020). Also, different components of BO can be transferred such as observations (Swersky et al., 2013), surrogate functions (Golovin et al., 2017; Wistuba and Grabocka, 2020), hyperparmater initializations (Wistuba and Grabocka, 2020), or all of them (Wei et al., 2021). However, most of the existing transfer-BO approaches assume the traditional black-box BO settings. To the best of our knowledge, Quick-Tune (Arango et al., 2023) is the only recent work which targets multi-fidelity and transfer BO at the same time. However, as mentioned above, their multi-fidelity BO formulation is greedy, whereas our transfer-BO method can dynamically control the degree of greediness during the BO by explicitly taking into consideration the trade-off between cost and performance of BO.

Cost-sensitive BO. Multi-fidelity BO is inherently cost-sensitive since predictions get more accurate as the gap between the fidelities becomes closer. However, the performance metric of such vanilla multi-fidelity BO monotonically increases as we spend more budget, whereas in this paper we want to find the optimal trade-off between the amount of budget spent thus far and the corresponding intermediate performances of BO, thereby automatically early-stopping the BO around the maximal utility. Quick-Tune (Arango et al., 2023) also suggests a cost-sensitive BO in multi-fidelity settings, but unlike our work, their primary focus is to trade-off between the performance and the cost of BO associated with pretrained models of various size, which can be seen as a generalization of more traditional notion of cost-sensitive BO (Snoek et al., 2012; Abdolshah et al., 2019; Lee et al., 2020), from black-box to multi-fidelity settings.

BO with user preference. Several works have tried to encode user's belief on good hyperparameter configurations into BO frameworks Souza et al. (2021); Hvarfner et al. (2021); Mallik et al. (2024). On the othet hand, our paper suggests encoding user's preference about the trade-off between cost and performance of multi-fidelity BO. Therefore, the notion of user preference in this paper is largely different from the previous literature.

B Details on Stop Criterion

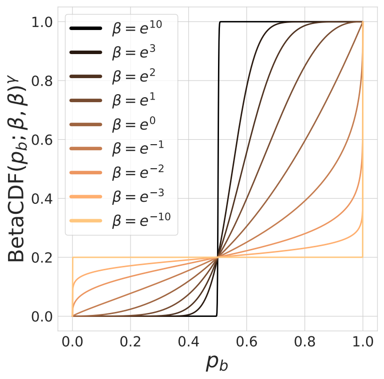


Figure 4: Eq. (3) with $\gamma = \log_2 5$ and the various values for β.

Fig. 4 plots $\mathrm{BetaCDF}(\cdot; \beta, \beta)^\gamma$ in Eq. (3) over the various values of β and when $\gamma = \log_2 5$. We see that the function becomes vertical as $\beta \rightarrow +\infty$ and horizontal as $\beta \rightarrow 0$. In the former case, we terminate the BO when $p_b < 0.5$ while ignoring the regret in the LHS of Eq. (2), whereas in the latter case we ignore p_b and only decide based on the regret, with the threshold δ_b fixed to a specific value specified by γ (e.g., $\gamma = \log_2 5$ corresponds to $\delta_b = 0.2$ in Fig. 4). Therefore, the role of β is to smoothly interpolate between the two extreme stopping criteria, whereas γ decides the range of the overall shape of the interpolated criterion.

C Details on Benchmarks and Data Preprocessing

In this section, we elaborate the details on the LC benchmarks and data preprocessing we have done.

LCBench. A LC dataset that evaluates the performance of 7 different hyperparameters on 35 different tabular datasets. The LCs are collected by training MLPs with 2,000 hyperparameter configurations, each for 51 epochs. We train our LC prediction model on 20 datasets and evaluate on the remaining 15 datasets. We use [APSFailure, Amazon_employee_access, Australian, Fashion-MNIST, KDDCup09_appetency, MiniBooNE, adult, airlines, albert, bank-marketing, blood-transfusion-service-center, car, christine, cnae-9, connect-4, covertype, credit-g, dionis, fabert, helena] for training LC extrapolator. We evaluate it on [higgs, jannis, jasmine, jungle_chess_2pcs_raw_endgame_complete, kc1, kr-vs-kp, mfeat-factors, nomao, numerai28.6, phoneme, segment, shuttle, sylvine, vehicle, volkert]. Each task contains 2000 LCs with 51 training epochs. We summarize the hyperparameter of LCBench in Table 2.

Table 2: The 7 hyperparameters for **LCBench** tasks.

Name	Type	Vaules	Info
batch_size	integer	$[16, 51]$	log
learning_rate	continuous	$[0.0001, 0.1]$	log
max_dropout	continuous	$[0.0, 1.0]$	
max_units	integer	$[64, 1024]$	log
momentum	continuous	$[0.1, 0.99]$	
max_layers	integer	$[1, 5]$	
weight_decay	continuous	$[1e-05, 0.1]$	

TaskSet. A LC dataset that consists of a diverse set of 1,000 optimization tasks drawn from various domains. We select 30 natural language processing (text classification and language modeling) tasks, train our LC extrapolator on 21 tasks, and evaluate on the remaining 9 tasks. Each task include 8 different hyperparameters and 1,000 their configurations. Each LC is collected by training models for 50 epochs. We use [rnn_text_classification_family_seed{19, 3, 46, 47, 59, 6, 66}, word_rnn_language_model_family_seed{22, 47, 48, 74, 76, 81}, char_rnn_language_model_family_seed{19, 26, 31, 42, 48, 5, 74}] for training LC extrapolator. We evaluate it on [rnn_text_classification_family_seed{8, 82, 89}, word_rnn_language_model_family_seed{84, 98, 99}, char_rnn_language_model_family_seed{84, 94, 96}]. Each task contains 1000 LCs with 50 training epochs. We summarize the hyperparameter of TaskSet in Table 3.

Table 3: The 8 hyperparameters for **Taskset** tasks.

Name	Type	Vaules	Info
learning_rate	continuous	$[1e-09, 10.0]$	log
beta1	continuous	$[0.0001, 1.0]$	
beta2	continuous	$[0.001, 1.0]$	
epsilon	continuous	$[1e-12, 1000]$	log
l1	continuous	$[1e-09, 10.0]$	log
l2	continuous	$[1e-09, 10.0]$	log
linear_decay	continuous	$[1e-08, 0.0001]$	log

PD1. A LC benchmark that includes the performance of modern neural architectures (including Transformers) run on large vision datasets such as CIFAR-10, CIFAR-100 Krizhevsky et al. (2009), ImageNet Russakovsky et al. (2015), as well as statistical modeling corpora and protein sequence datasets from bioinformatics. We select 23 tasks with 4 different hyperparameters based on SyneTune Salinas et al. (2022) package, train our LC extrapolator on 16 tasks, and evaluate on the remaining 7 tasks. For easier transfer learning, we preprocess the data by excluding hyperparameter configurations with their training diverging (e.g., LCs with `NaN`), and linearly interpolate the LCs to match their length across different tasks. We then obtain the LCs of 50 epochs over the 240 configurations. We use [uniref50_transformer_batch_size_128, lm1b_transformer_batch_size_2048,
imagenet_resnet_batch_size_256, mnist_max_pooling_cnn_tanh_batch_size_2048,

mnist_max_pooling_cnn_relu_batch_size_{2048, 256}, mnist_simple_cnn_batch_size_{2048, 256}, fashion_mnist_max_pooling_cnn_tanh_batch_size_2048, fashion_mnist_max_pooling_cnn_relu_batch_size_{2048, 256}, fashion_mnist_simple_cnn_batch_size_{2048, 256}, svhn_no_extra_wide_resnet_batch_size_1024, cifar{100, 10}_wide_resnet_batch_size_2048] for training LC extrapolator. We evaluate it on [imagenet_resnet_batch_size_512, translate_wmt_xformer_translate_batch_size_64, mnist_max_pooling_cnn_tanh_batch_size_256, fashion_mnist_max_pooling_cnn_tanh_batch_size_256, svhn_no_extra_wide_resnet_batch_size_256, cifar100_wide_resnet_batch_size_256, cifar10_wide_resnet_batch_size_256]. Each task contains 240 LCs with 50 training epochs. We summarize the hyperparameter of PD1 in Table 4.

Table 4: The 8 hyperparameters for **PD1** tasks.

Name	Type	Vaules	Info
lr_initial_value	continuous	$[1e-05, 10.0]$	log
lr_power	continuous	$[0.1, 2.0]$	
lr_decay_steps_factor	continuous	$[0.01, 0.99]$	
one_minus_momentum	continuous	$[1e-05, 1.0]$	log

Data Preprocessing. As will be detailed in the §G, we use the 0-epoch LC value $y_{n,0}$ which is the performance before taking any gradient steps. The 0-epoch LC values originally are not provided except for LCBench; we use the log-loss of the first epoch as the 0-epoch LC value for TaskSet, as it is already sufficiently large in our chosen tasks. For PD1, we interpolate the LCs to be the length of 51 training epochs, and we take the first performance as the 0-epoch LC value. Furthermore, we take the average over the 0-epoch LC values $\bar{y}_0$ since it is hard to have different initial values among optimizer hyperparameter configurations in a task, without taking any gradient steps. For transfer learning, we follow the convention of PFN Adriaensen et al. (2023) for data preprocessing; we consistently apply non-linear LC normalization[2] to the LC data of three benchmarks, which not only maps either accuracy or log-loss LCs into $[0, 1]$ but also simply make our optimization as a maiximization problem. To facilitate transfer learning, we use the maximum and minimum values in each task in LCBench and PD1 benchmark for the LC normalization. In TaskSet, we only use the $\bar{y}_0$ for the LC normalization.

D Transfer learning with LC mixup.

LC Mixup. Among many plausible options, in this paper we propose to use Prior Fitted Networks (PFNs) (Müller et al., 2021) for LC extrapolation. PFNs are an in-context Bayesian inference method based on Transformer architectures (Vaswani et al., 2017), and show good performances on LC extrapolation (Adriaensen et al., 2023; Rakotoarison et al., 2024) without the computationally expensive online retraining Kadra et al. (2023). A major difficulty of using PFNs for our purpose is that they are not trained with the existing datasets, but trained with a prior distribution, which is essential to perform Bayesian inference. Also, PFNs require relatively a large Transformer architecture (Vaswani et al., 2017) as well as huge amounts of training examples for good generalization performance (Adriaensen et al., 2023), which makes it risky to train PFNs with a finite set of examples.

Here we explain our novel transfer learning method for PFNs that can circumvent those difficulties with the mixup strategy (Zhang et al., 2018). Suppose we have M different datasets and the corresponding M sets of LCs collected from N hyperparameter configurations. Define $l_{m,n} = (y^m_{n,1}, \dots, y^m_{n,T})$, the T-dimensional row vector of validation performances (y's) collected from the m-th dataset and the n-th configuration, forming a complete LC of length T. Further define the matrix $L_m = [l^\top_{m,1}; \dots; l^\top_{m,N}]^\top$, the row-wise stack of those LCs. In order to augment training examples, we propose to perform the following two consecutive mixups (Zhang et al., 2018):

1. Across datasets: $L' = \lambda_1 L_m + (1-\lambda_1)L_{m'}$, with $\lambda_1 \sim \text{Unif}(0,1)$, for all $m, m' \in [M]$.

[2] The details can be found in Appendix A of PFN (Adriaensen et al., 2023) and `https://github.com/automl/lcpfn/blob/main/lcpfn/utils.py`.

2. Across configurations: $(x'', l'') = \lambda_2(x_n, l'_n) + (1-\lambda_2)(x_{n'}, l'_{n'})$
   with l'_n the n-th row of L', $\lambda_2 \sim \text{Unif}(0,1)$, for all L' and $n, n' \in [N]$.

In this way, we can sample infinitely many training examples $\{(x'', l'')\}$ by interpolating between the LCs, leading to a robust LC extrapolator with less overfitting. Note that in the first step, we do not individually perform the mixup over the configurations but apply the same λ_1 to all the configurations, in order to preserve the correlations between the configurations encoded in the given datasets.

Connection to ifBO and Neural Process. Our mixup strategy is reminiscent of the data generation scheme of ifBO (Rakotoarison et al., 2024), a variant of PFNs for in-context freeze-thaw BO. Similarly to our ancestral sampling, ifBO first samples random weights for a neural network (i.e., a prior distribution) to sample a correlation between configurations (the first mixup step), and then linearly combines a set of basis functions to generate LCs (the second mixup step). Our training method differs from ifBO in that our prior distribution is implicitly defined by LC datasets and the mixup strategy, whereas ifBO resorts to a manually defined distribution.

Indeed, our training method is more similar to Transformer Neural Processes (TNPs) (Nguyen and Grover, 2022), a Transformer variant of Neural Processes (NPs) (Garnelo et al., 2018). Similarly to PFNs, TNPs directly maximize the likelihood of target data given context data with a Transformer architecture, which differs from the typical NP variants that summarize the context into a latent variable and perform variational inference on it. Moreover, as with the other NP variants, TNPs meta-learn a model over a distribution of tasks to perform sample efficient probabilistic inference. In this vein, the whole training pipeline of our LC extrapolator can be seen as an instance of TNPs – we also meta-learn a sample efficient Transformer-based LC extrapolator over the distribution of LCs induced by the mixup strategy.

E Details on Architecture and Training of LC Extrapolator

In the section, we elaborate our LC extrapolator model and how to train it on the learning curve dataset.

Construction of Context and Query points.. The whole training pipeline of our learning curve extrapolator model can be seen an instance of TNPs (Nguyen and Grover, 2022). Here we can simulate each step of Bayesian Optimization; predicting the remaining part of LC in all configurations conditioned on the set $\mathcal{C}$ of the collected partial LCs. To do so, we construct a training task by randomly sampling context and query points from LC benchmark after the proposed LC mixup as follows:

1. We choose a LC dataset $L_m = [l_{m,1}^\top; \dots; l_{m,N}^\top]^\top \in \mathbb{R}^{N\times T}$ by randomly sampling $m \in [M]$.
2. From L_m, we randomly sample $n_1, \dots, n_C \in [N]$ and $t_1, ..., t_C \in [T]$ and construct context points of $X^{(c)} = [x_{n_1}^\top, \dots, x_{n_C}^\top]^\top \in \mathbb{R}^{C\times d_x}$, $T^{(c)} = [t_1/T, \dots, t_C/T]^\top \in \mathbb{R}^{C\times 1}$, and $Y^{(c)} = [y_{n_1,t_1}, \dots, y_{n_C,t_C}] \in \mathbb{R}^{C\times 1}$.
3. From L_m, we exclude $n_1, \dots, n_C \in [N]$ and $t_1, ..., t_C \in [T]$ and randomly sample $n'_1, \dots, n'_Q \in [N]$ and $t'_1, ..., t'_Q \in [T]$ and construct query points of $X^{(q)} = [x_{n'_1}^\top, \dots, x_{n'_Q}^\top]^\top \in \mathbb{R}^{Q\times d_x}$, $T^{(q)} = [t'_1/T, \dots, t'_C/T]^\top \in \mathbb{R}^{Q\times 1}$, and $Y^{(q)} = [y_{n'_1,t'_1}, \dots, y_{n'_Q,t'_Q}] \in \mathbb{R}^{Q\times 1}$.

Transformer for Predicting Learning Curves.. From now on, we denote each row vector of the constructed context and query points with the lowercase, e.g., $y^{(q)}$ of $Y^{(q)}$. We learn a Transformer-based learning curve extrapolator model which is a probabilistic model of $f(Y^{(q)}|X^{(c)}, T^{(c)}, Y^{(c)}, X^{(q)}, T^{(q)})$. Conditioned on any subsets of LCs (i.e., $X^{(c)}$, $T^{(c)}$, and $Y^{(c)}$), this model predicts a mini-batch of the remaining part of LCs of existing hyperparameter configurations in a given dataset (i.e., $Y^{(q)}$ of $X^{(q)}$ and $T^{(q)}$). For the computational efficiency, we further assume that the query points are independent to each other, as done in PFN (Adriaensen et al., 2023):

$$f(Y^{(q)}|X^{(c)}, T^{(c)}, Y^{(c)}, X^{(q)}, T^{(q)}) = \prod_{x^{(q)}, t^{(q)}, y^{(q)}} f(y^{(q)}|x^{(q)}, t^{(q)}, X^{(c)}, T^{(c)}, Y^{(c)}). \tag{5}$$

Before encoding the input into the Transformer, we first encode the input of $X^{(c)}, T^{(c)}, Y^{(c)}, X^{(q)}$, and $T^{(q)}$ using simple linear layer as follows:

$$H^{(c)} = X^{(c)} W_x + T^{(c)} W_t + Y^{(c)} W_y \tag{6}$$

$$H^{(q)} = X^{(q)} W_x + T^{(q)} W_t, \tag{7}$$

where $W_x \in \mathbb{R}^{d_x \times d_h}$, $W_t \in \mathbb{R}^{1 \times d_h}$, and $W_y \in \mathbb{R}^{1 \times d_h}$. Here, we abbreviate the bias term.

Then we concatenate the encoded representations of $H^{(c)}$ and $H^{(q)}$, and feedforward it into Transformer layer by treating each pair of each row vector of $H^{(c)}$ and $H^{(q)}$ as a separate position/token as follows:

$$H = \texttt{Transformer}([H^{(c)}; H^{(q)}, \texttt{Mask}]) \in \mathbb{R}^{(M+N) \times d_h} \tag{8}$$

$$\hat{Y} = \texttt{Head}(H) \in \mathbb{R}^{(M+N) \times d_o}, \tag{9}$$

where $\texttt{Transformer}(\cdot)$ and $\texttt{Head}(\cdot)$ denote the Transformer layer and multi-layer perceptron (MLP) for the output prediction, respectively. $\texttt{Mask} \in \mathbb{R}^{(N_c+N_q)\times(N_c+N_q)}$ is the mask of transformer that allows all the tokens to attend context tokens only. Here, the output dimension d_o is specified by output distribution of y. Following PFN (Adriaensen et al., 2023), we discretize the domain of y by $d_o = 1000$ and use the categorical distribution. Finally, we only take the output of the last N_q tokens as output, i.e., $\hat{Y}^{(q)} = \hat{Y}[:, N_c : (N_c + N_q)] \in \mathbb{R}^{N_q \times d_h}$ (PyTorch-style indexing operation), since we only need the outputs of query tokens for modeling $\prod f(y^{(q)} | x^{(q)}, t^{(q)}, X^{(c)}, T^{(c)}, Y^{(c)})$.

Training Objective.. Our pre-training objective is then defined as follows:

$$\arg\min_f \mathcal{E}_p \left[- \sum_{x^{(q)}, t^{(q)}, y^{(q)}} \log f(y^{(q)} | x^{(q)}, t^{(q)}, X^{(c)}, T^{(c)}, Y^{(c)}) \right] + \lambda_{\text{PFN}} \mathcal{L}_{\text{PFN}}, \tag{10}$$

where $\mathcal{D}_{KL}$ is the Kullback–Leibler divergence, and p is the empirical LC data distribution. We additionally minimize $\mathcal{L}_{\text{PFN}}$ with coefficient λ_{PFN}, which is the LC extrapolation loss in each LC (Adriaensen et al., 2023). We found $\lambda_{\text{PFN}} = 0.1$ works well for most cases. We use the stochastic gradient descent algorithm to solve the above optimization problem.

Training Details.. We sample 4 training tasks for each iteration, i.e., the size of meta mini-batch is set to 4. We uniformly sample the size C of context points from 1 to 300, and the size of query points Q is set to 2048. Following PFN (Adriaensen et al., 2023), the hidden size of each Transformer block d_h, the hidden size of feed-forward networks, the number of layers of Transformer, dropout rate are set 1024, 2048, 12, 0.2. We use GeLU (Hendrycks and Gimpel, 2016). We train the extrapolator for 10,000 iterations on training split of each benchmark with Adam Kingma and Ba (2014) optimizer. The ℓ_2 norm of meta mini-batch gradient is clipped to 1.0. The learning rate is linearly increased to 2e-05 for 25000 iterations, and it is decreased with a cosine scheduling until the end. The whole training process takes roughly 10 hours in one NVIDIA Tensor Core A100 GPU.

F Baseline

We list the implementation details for baselines as follows:

1. **Random Search**. (Bergstra and Bengio, 2012) Instead of randomly selecting a hyperparameter configuration for each BO step, we run the selected configuration until the last epoch T.
2. **ASHA, BOHB, and DEHB**. We next compare against several variants of Hyperband (Li et al., 2018) such as **ASHA** (Li et al., 2020a) the asynchronous parallel version of it, **BOHB** (Falkner et al., 2018) which replaces its random sampling of configurations with BO, and **DEHB** (Awad et al., 2021) which promotes internal knowledge transfer with evolution strategy. We follow the most recent implementation of these algorithms in Quick-Tune (Arango et al., 2023). We slightly modify the official code[3], which is heavily based on SyneTune (Salinas et al., 2022) package.

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

3. **DyHPO.** We also compare against more recent multi-fidelity BO methods such as **DyHPO** (Wistuba et al., 2022) which uses deep kernel GP (Wilson et al., 2016) and a greedy acquisition function with a short-horizon LC extrapolation. We follow the official code[4] provided the authors of DyHPO Wistuba et al. (2022), and slightly modify the benchmark implementation to incorporate our experimental setups.
4. **Quick-Tune†.** This is a modified version of Quick-Tune (Arango et al., 2023) which is originally developed for dynamically selecting both pretrained models and hyperparmater configurations, with the additional cost term penalizing the non-uniform evaluation wall-time associated with each joint configuration. Since our experimental setup does not consider selecting pretrained models nor non-uniform evaluation wall-time, we only leave the transfer learning part of the model, which corresponds to a transfer learning version of DyHPO, i.e., we train its surrogate function with the same LC datasets used for training our LC extrapolator. For Quick-Tune†, we pretrain the deep kernel GP for 50000 iterations with Adam optimizer with mini-batch size of 512. The initial learning rate is set to 1e-03 and decayed with cosine scheduling. To leverage the transfer learning scenario, we use the best configuration among the LC datasets which is used for training the GP as an initial guess of BO.
5. **DPL.** This method extrapolates LCs with power law functions and model ensemble. We follow the official code[5] provided the authors of DPL (Kadra et al., 2023), and slightly modify the benchmark implementation to incorporate our experimental setups.
6. **FSBO.** This is a black-box transfer-BO method that uses the same LC datasets to train a deep kernel GP surrogate function. The difference of FSBO from Quick-Tune† is that its surrogate models the validation performances at the last epoch, whereas the surrogate of Quick-Tune† predicts the performances at the next epoch for multi-fidelity HPO. FSBO does not provide an official code, therefore, we follow an available code in the internet[6]. We also slightly modify the benchmark implementation, and use the best configuration among the LC datasets as an initial guess.

G Experimental Setups

In this section, we elaborate details on the experimental setups.

Utility function. While there are many plausible options for the utility function, in this paper we use a linear function for penalizing the cost of multi-fidelity BO, i.e., $U(b, \tilde{y}) = \tilde{y} - \alpha b$ where $\tilde{y}$ is the BO performance, b the BO steps, and $\alpha \in \{0, 4e\text{-}05, 2e\text{-}04\}$. Note that $\alpha = 0$ does not penalize the number of BO steps at all, hence the BO does not terminate until the last BO step B as with the conventional multi-fidelity BO setup.

Evaluation metric. In order to report the average performances over the tasks, we use the normalized regret of utility $(U_{\max} - U_{b^*})/(U_{\max} - U_{\min}) \in [0, 1]$, similarly to Eq. (2). U_{b^*} is the utility obtained right after the BO terminates at step b^*, and $U_{\max}$ is the maximum achievable by running a single optimal configuration up to its maximum utility. Computing the exact $U_{\min}$ is a difficult combinatorial optimization problem, thus we simply approximate it with $U(B, y_1^{\text{worst}})$, where y_1^{worst} is the worst 1-epoch validation performance across the configurations – we simply let y_1^{worst} decay over the maximum BO steps B, corresponding to a lower bound of the exact $U_{\min}$. We then average the normalized regret across all the tasks in each benchmark, and report the mean and standard deviation over 5 runs. Lastly, we also report the rank of each method averaged over the tasks.

0-epoch LC value. We assume the access of the 0-epoch LC value $\bar{y}_0$ in §C which is the model performance before taking gradient steps. This is also plausible for realistic scenarios since in most deep-learning models one evaluation cost is acceptable in comparison to training costs. The 0-epoch LC value $\bar{y}_0$ is always conditioned on our LC extrapolator f for both pretraining and BO stage.

Monte-Carlo (MC) sampling for reducing variance of LCs. As mentioned in §2.1, we estimate the expectation of proposed acquisition function A in Eq. (1) with 1000 MC samples. We found that each LC $y_{n,t_{n:T}}$ sampled from LC extrapolator $f(\cdot|x_n, \mathcal{C})$ is noisy, due to the assumption that query points of $y_{n,t_{n:T}}$ are

[4]https://github.com/releaunifreiburg/DyHPO

[5]https://github.com/releaunifreiburg/DPL

[6]https://github.com/releaunifreiburg/fsbo

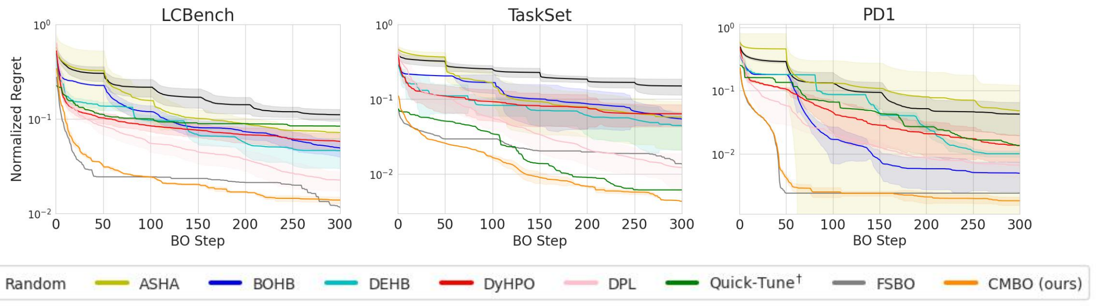
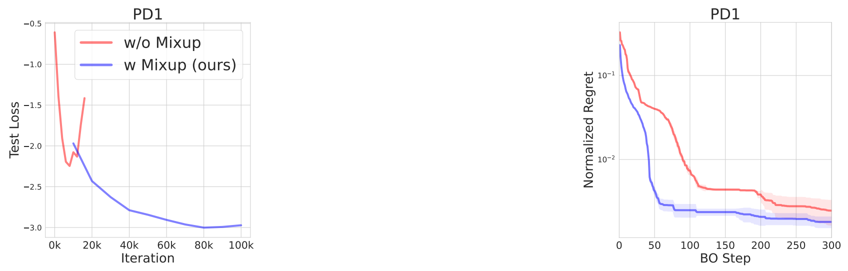


Figure 5: **The results on the conventional multi-fidelity HPO setup** ($\alpha = 0$). For each benchmark, we report the normalized regret of utility aggregated over all the test datasets.



(a) Test loss (b) Normalized Regret

Figure 6: Ablation study on the mixup training. We use $\alpha = 0$ and PD1 benchmark for the experiments.

independent to each other in Eq. (5). We compute $\tilde{y}_{b+\Delta t}$ by taking the maximum among the last step BO performance (i.e., cumulative max operation), therefore, the quality of estimation highly degenerates due to the noise in the small Δt. To prevent this, we reduce the variance of MC samples by taking the average of the sampled LCs. For example, we sample 5000 LC samples from the LC extrapolator f, then we divide them into 1000 groups and take the average among the 5 LC samples in each group. We empirically found that this stabilize the estimation of not only acquisition function A and probability of utility improvement p_b in Eq. (4).

Inference Time for BO. The most of time for each BO step in our method is spent during LC extrapolation. In Table 5, we report the wall-clock time spent on LC extrapolation for 100 mini-batches of LCs. The wall-clock times vary depending on the context size. We measure all the wall-clock times in one one NVIDIA Tensor Core A100 GPU.

Table 5: **Wall-clock time for Inference** on 100 mini-batches of LCs.

Context Size	Inference Time (s)
1	0.00921010971069336
10	0.01493692398071289
20	0.01413583755493164
50	0.017796993255615234
100	0.01770782470703125
200	0.025087356567382812
300	0.027765989303588867

H Additional Experimental Results

Effectiveness of our transfer learning.. We first demonstrate the effectiveness of our transfer learning method. Fig. 5 shows the results on the conventional multi-fidelity HPO setting where we do not penalize the cost of BO at all ($\alpha = 0$). First of all, note that FSBO, a black-box transfer-BO method which switches its configuration only after a single complete training (e.g., 50 epochs), even outperforms all the other multi-fidelity methods that can change the configurations every epoch. The result clearly shows the importance of transfer learning for improving the sample efficiency of HPO. Quick-Tune†, a transfer version of DyHPO, performs similarly to the other baselines despite of the transfer learning, except on TaskSet benchmark. We attribute this result to its greedy acquisition function, and more importantly its lack of data augmentation. On the other hand, our method is non-greedy (when $\alpha = 0$) and can effectively augment the data with our mixup strategy, thereby showing significantly better performances than all the other multi-fidelity methods. Fig. 6 shows the ablation study on our mixup training. Fig. 6a shows that we can effectively reduce the risk of overfitting by adding the mixup strategy. As a result, the performance of BO improves significantly (Fig. 6b). Lastly, our method significantly outperforms FSBO on TaskSet and slightly on LCBench and PD1, showing the superiority of multi-fidelity BO to black-box BO.

Visualizations of the normalized regret over BO steps. for LCBench ($\alpha = 4e$-05), LCBench ($\alpha = 2e$-04), TaskSet ($\alpha = 4e$-05), TaskSet ($\alpha = 2e$-04), PD1 ($\alpha = 4e$-05), and PD1 ($\alpha = 2e$-04) are provided Figure 7, 8, 9, 10, 11, and 12, respectively. These figures illustrate the normalized regret over the course of BO, where our method achieves significantly lower regret upon termination. Our method tends to achieve the minimum regret earlier than the baselines, demonstrating its sample efficiency in searching good hyperparameter configurations by explicitly considering the utility during the BO.

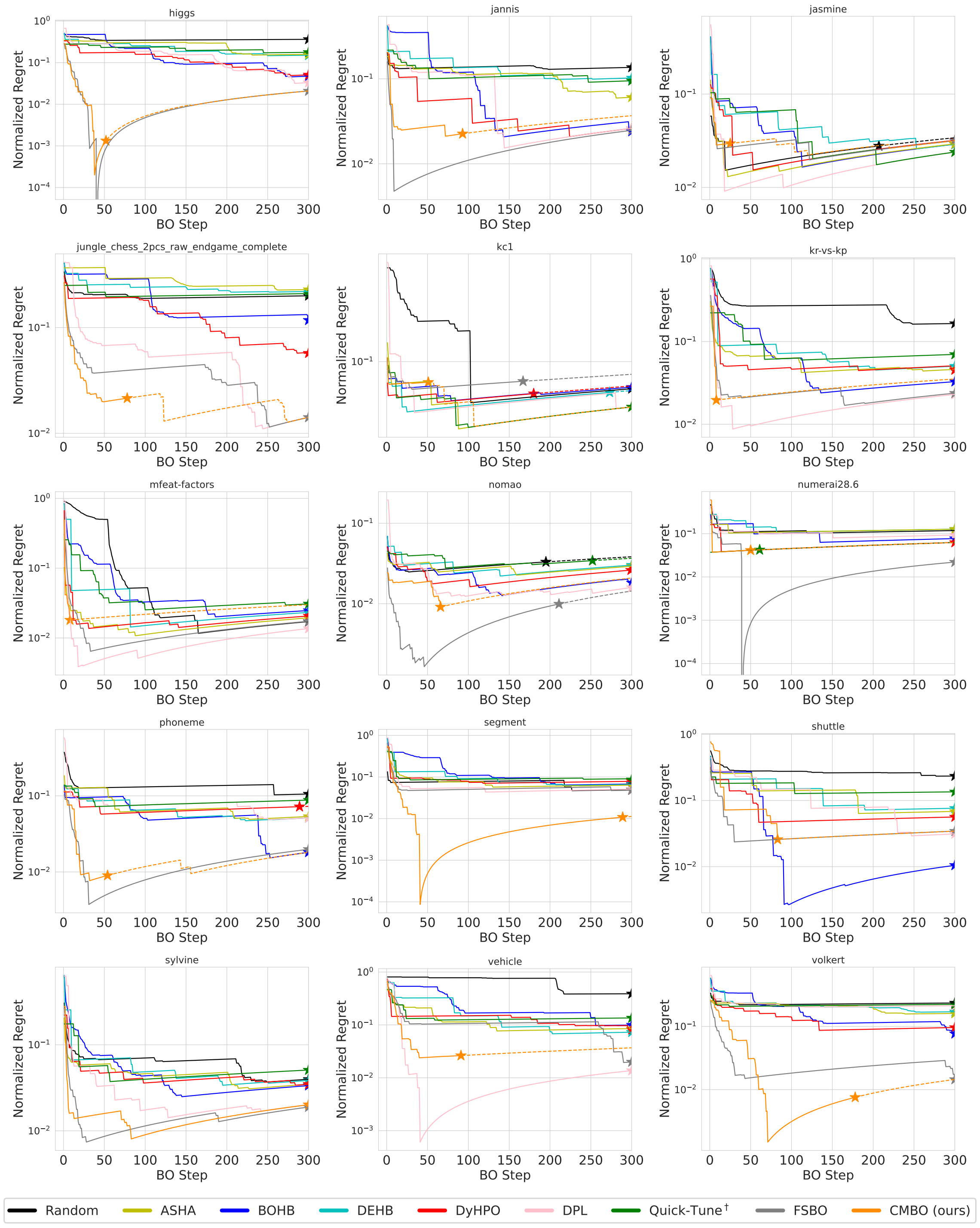


Figure 7: Visualization of the normalized regret over BO steps on **LCBench** (α =**4e-05**).

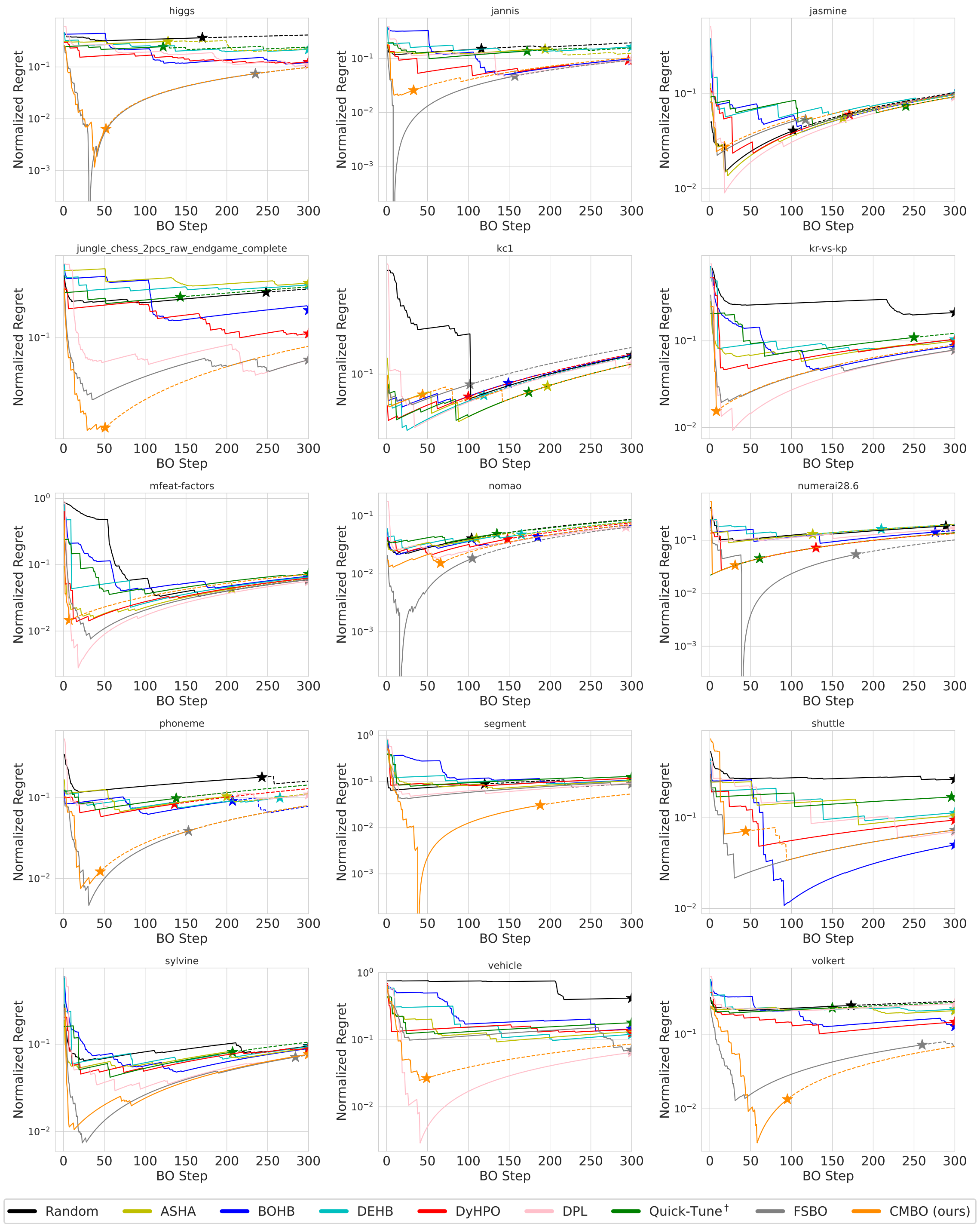


Figure 8: Visualization of the normalized regret over BO steps on **LCBench (α =2e-04)**.

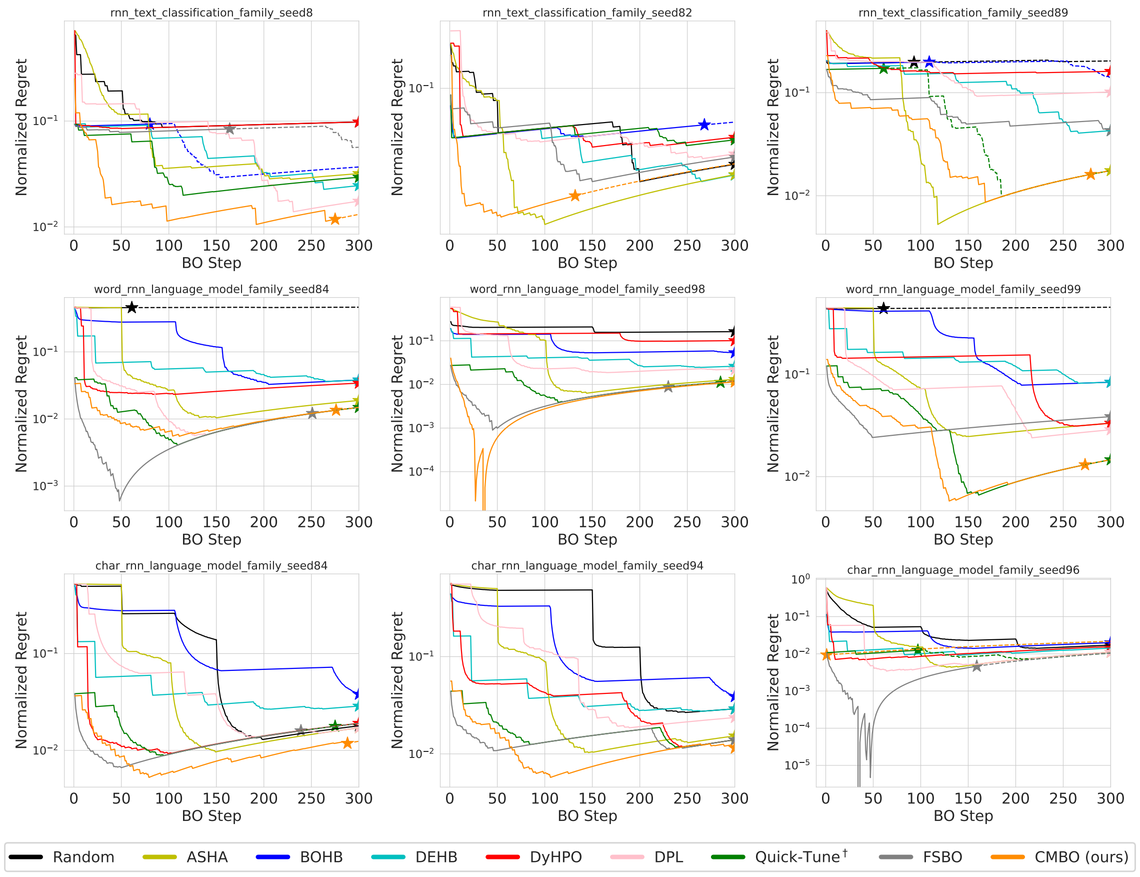


Figure 9: Visualization of the normalized regret over BO steps on **TaskSet (α =4e-05)**.

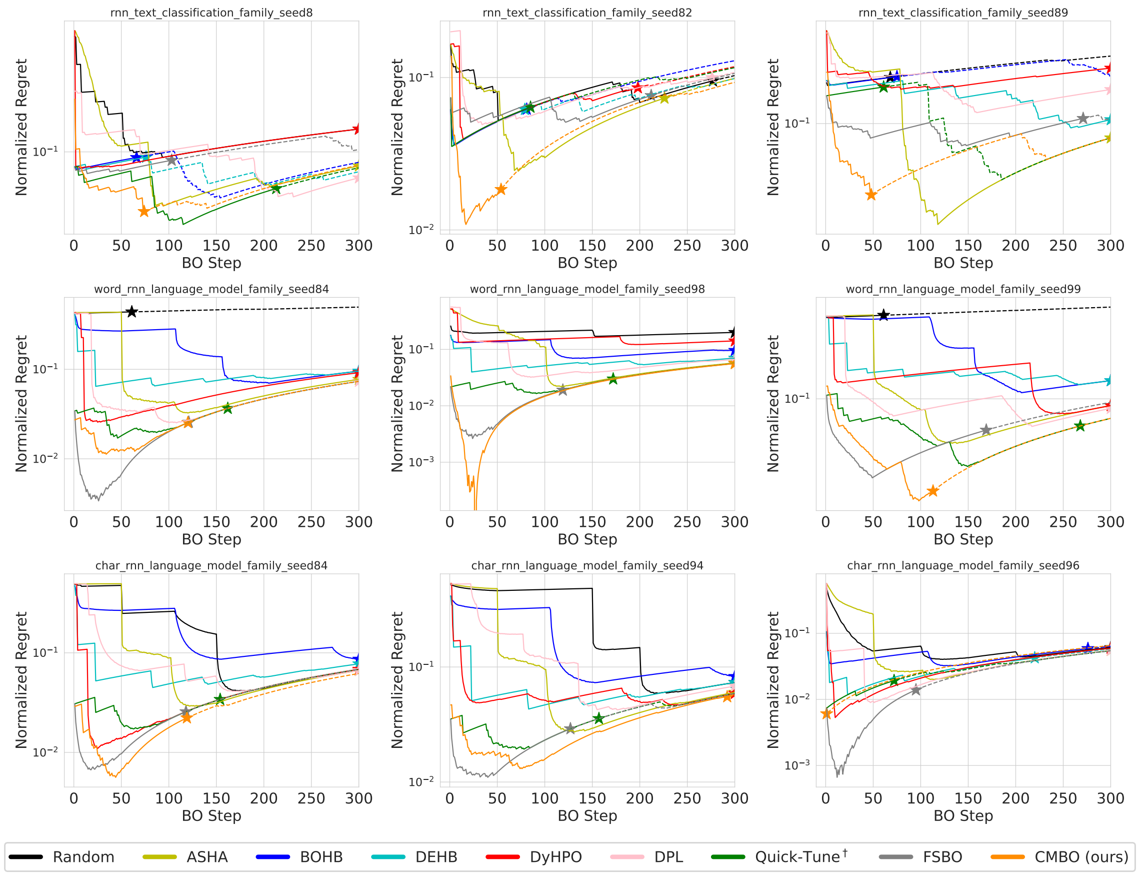


Figure 10: Visualization of the normalized regret over BO steps on **TaskSet (α =2e-04)**.

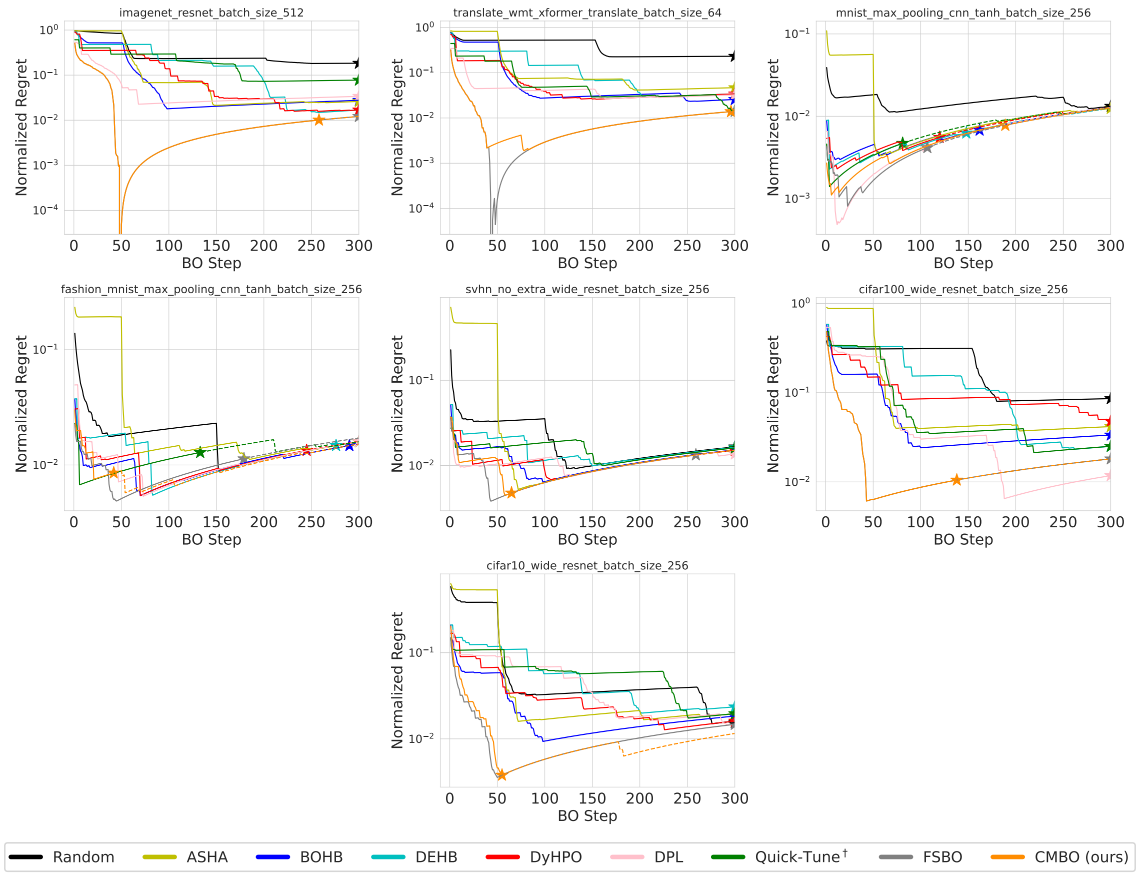


Figure 11: Visualization of the normalized regret over BO steps on **PD1 (α =4e-05)**.

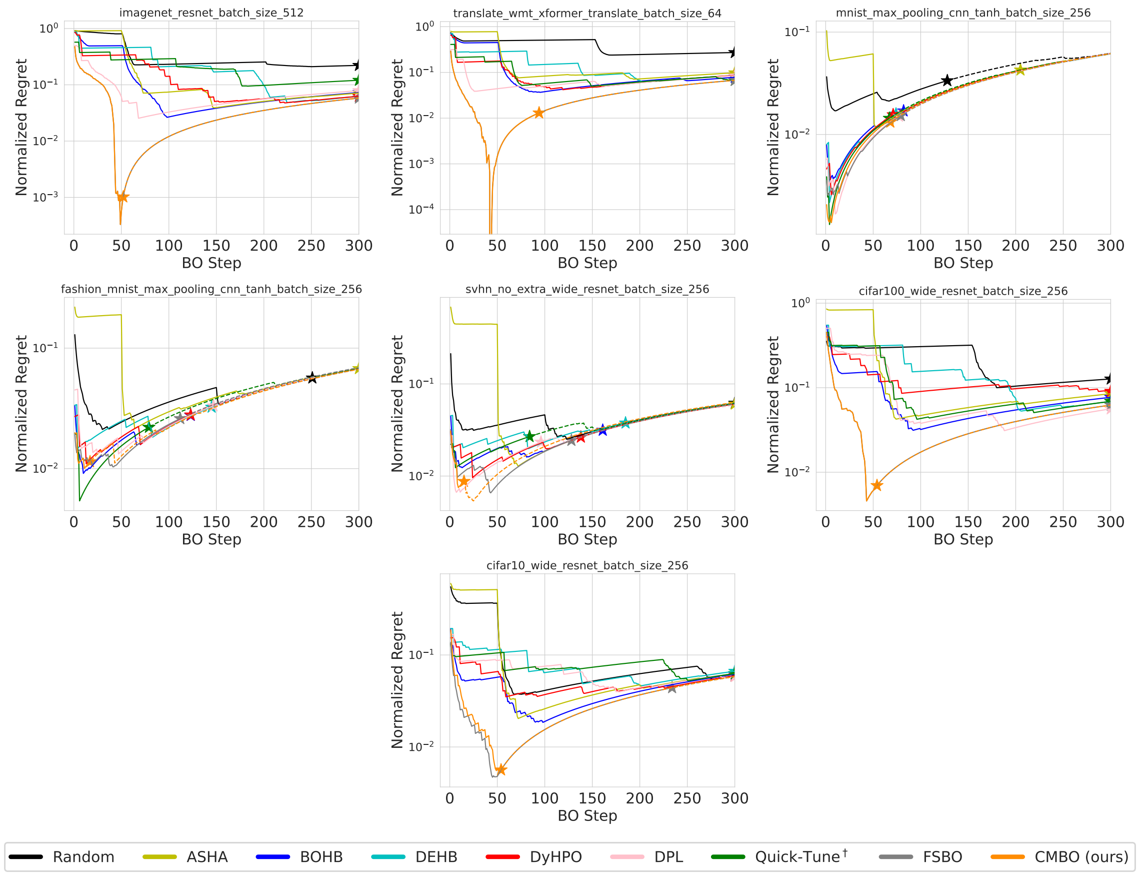


Figure 12: Visualization of the normalized regret over BO steps on **PD1 (α =2e-04)**.

Visualizations of the LC extrapolation over BO steps. for LCBench, TaskSet, and PD1 are provided Figure 13, 14, and 15, respectively. Here, we plot the LC extrapolation results of unseen hyperparameter configurations through BO. Each row shows the results for a different size of the observation set ($|\mathcal{C}| = 0, 10, 50,$ and 300), and each column shows a different size of context points in each LC (0, 2, 5, 10, 20, and 30).

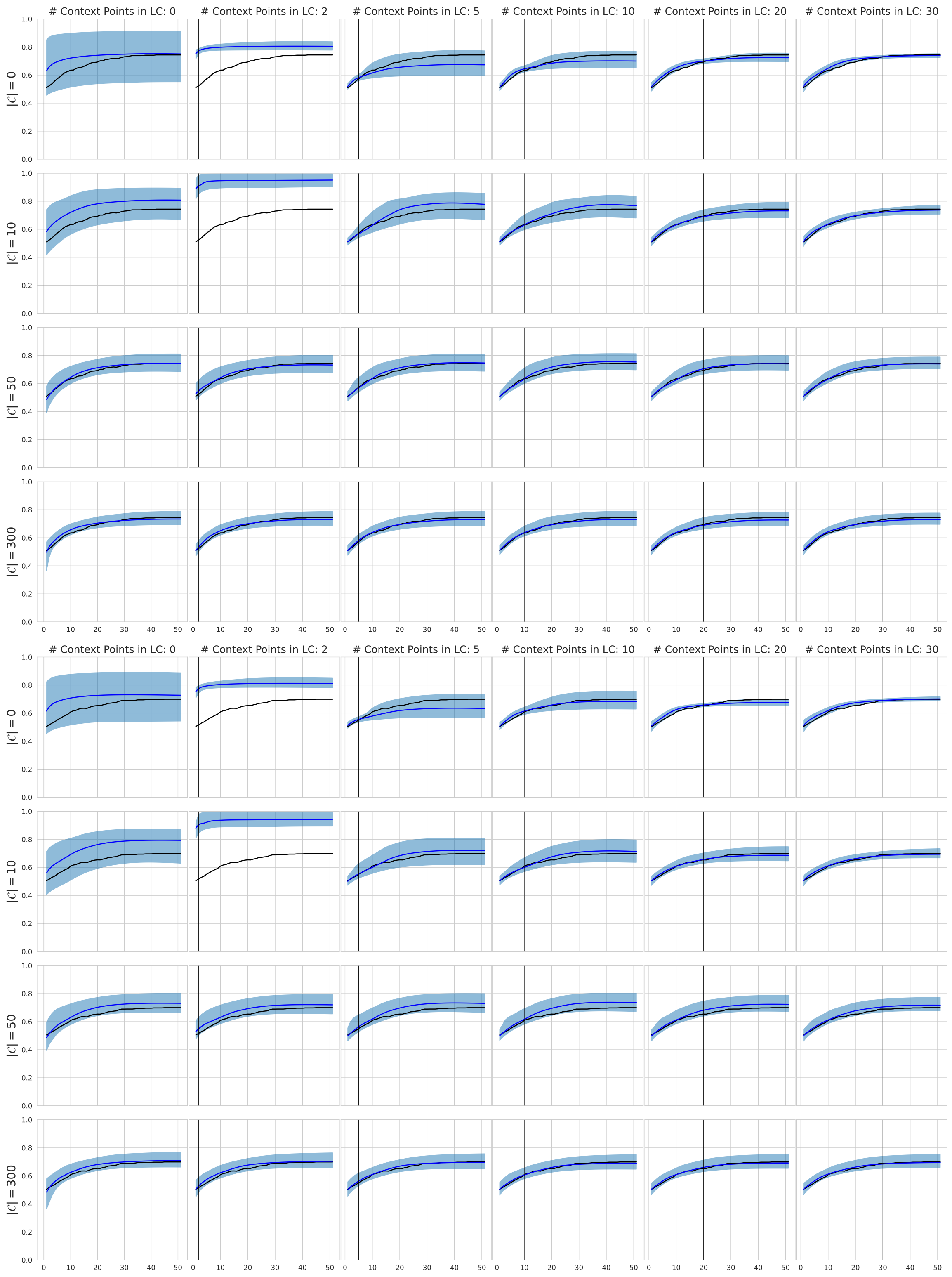
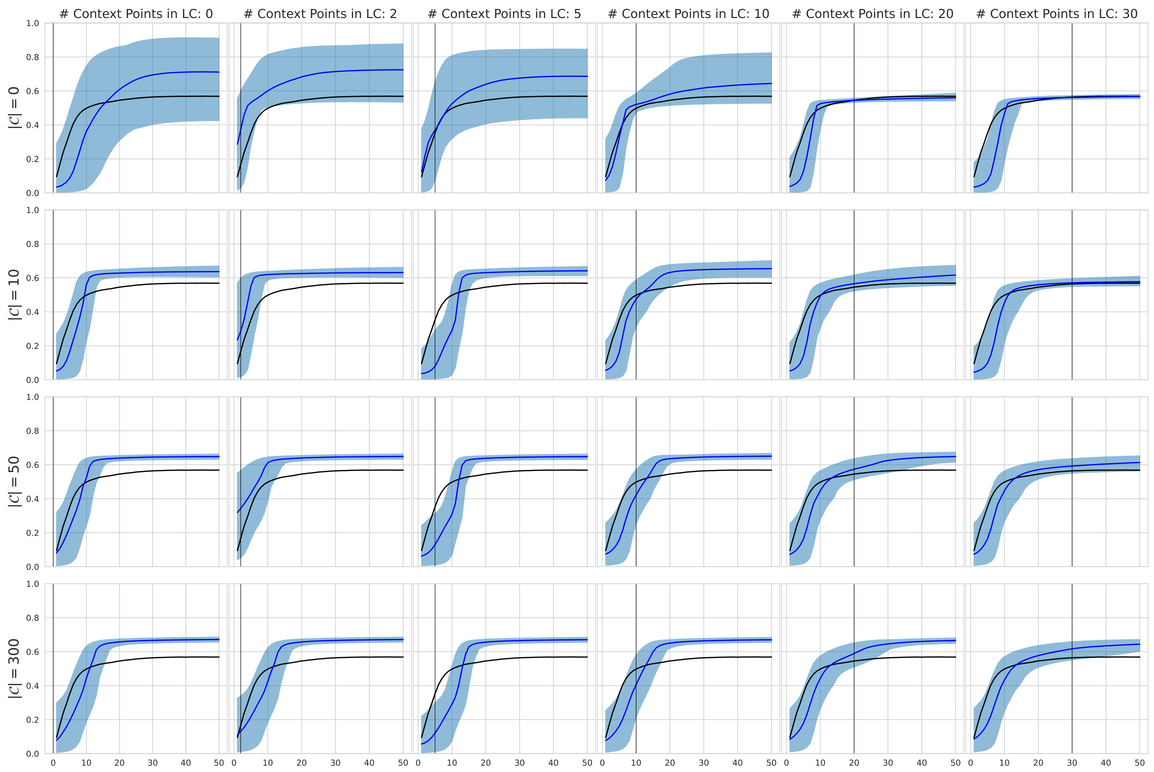


Figure 13: Visualization of LC extrapolation over BO steps on **LCBench**.



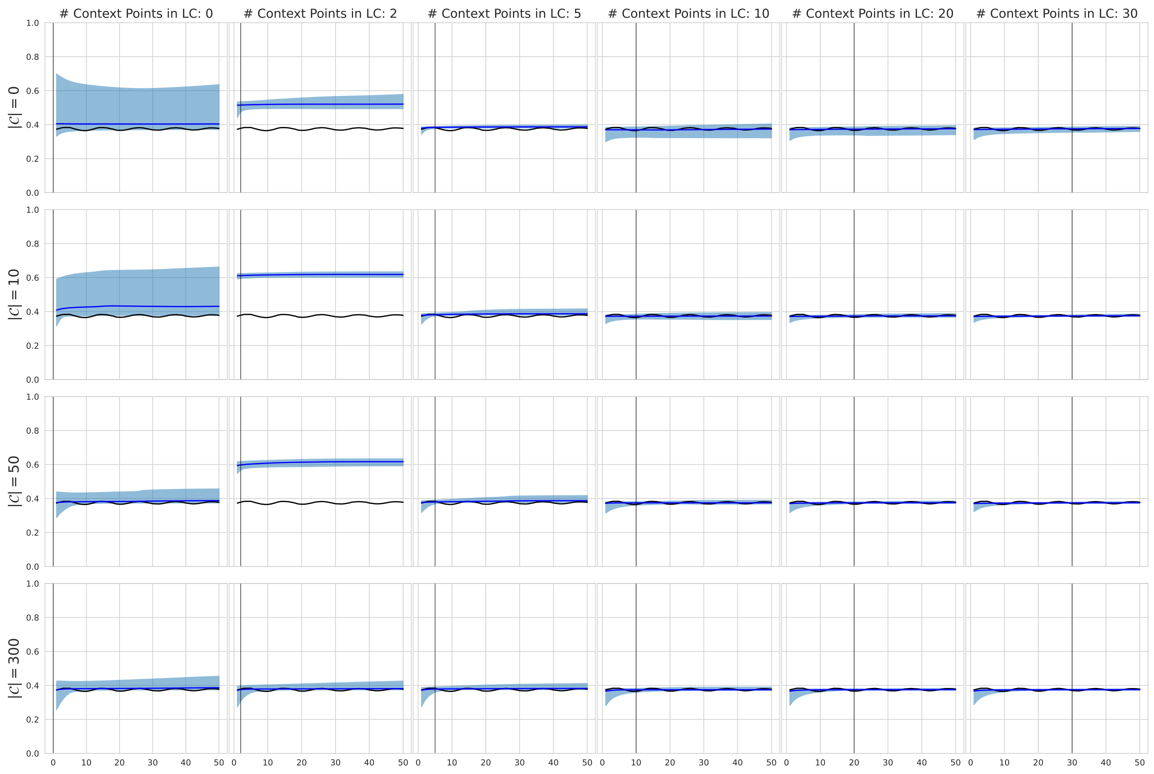


Figure 14: Visualization of LC extrapolation over BO steps on **TaskSet**.

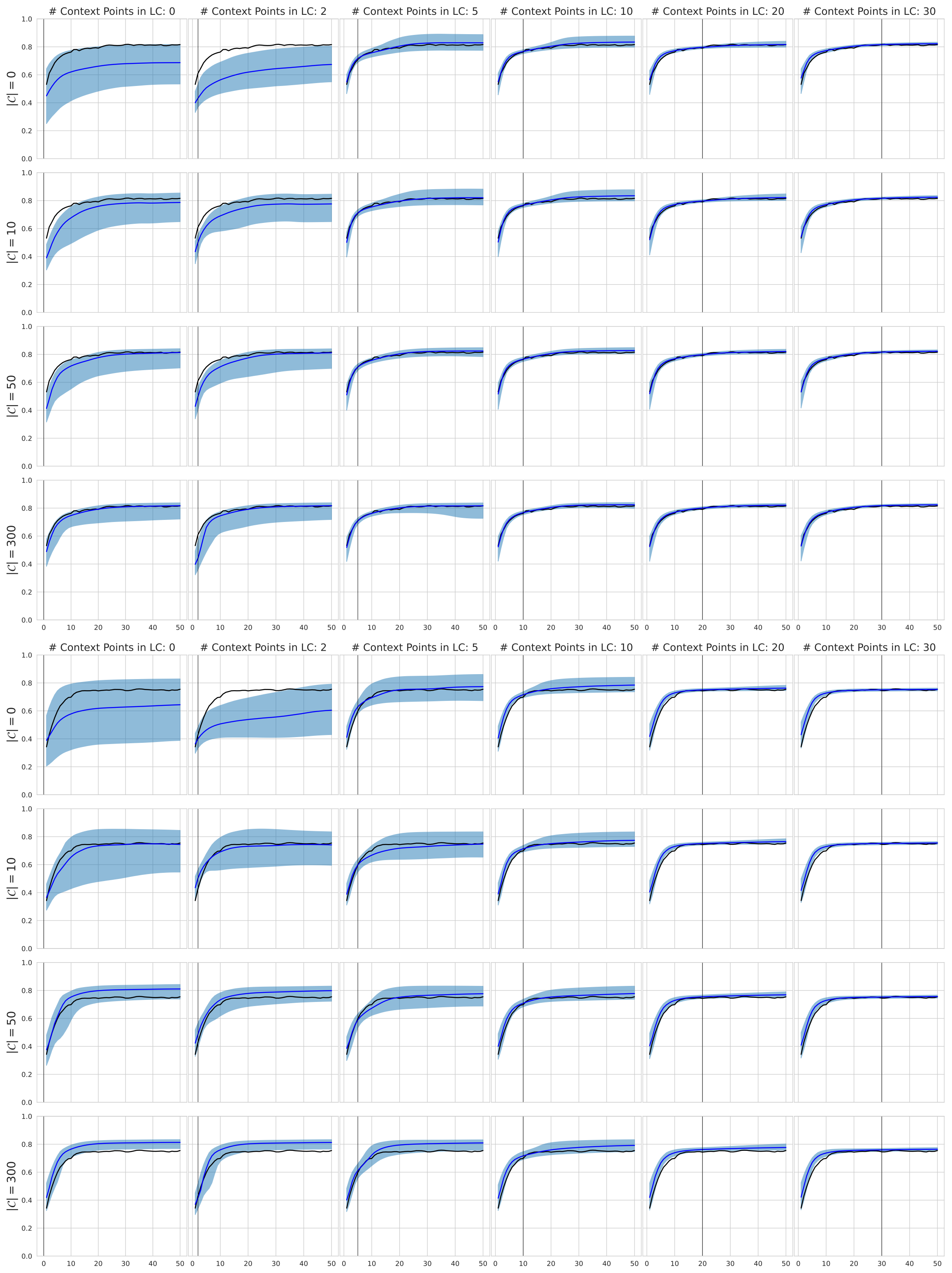


Figure 15: Visualization of LC extrapolation over BO steps on **PD1**.