Efficient Hyperparameter Optimization with Adaptive Fidelity Identification

Supplementary Material

7. Notation

In Table 3, we provide a comprehensive summary of the notations utilized throughout the paper, along with their detailed definitions and explanations.

8. Proof of FastBO

HPO methods generally do not provide theoretical guarantees or rely on strong assumptions. In § 4.1, we provide formal definitions for the efficient point and propose to use the efficient point e_i of $\boldsymbol{\lambda}_i$ as its fidelity to fit the surrogate model. While it is challenging to prove FastBO's efficiency in reaching the optimal configuration, we provide a proof showing the superiority of FastBO over SHA-based methods (*e.g.*, Hyperband, ASHA, PASHA, BOHB, A-BOHB, A-CQR, Hyper-Tune). We show that fidelities in FastBO more reliably indicate final fidelity performance than those in SHA-based methods.

Proof. Given two learning curves $\mathcal{C}_1(r)$, $\mathcal{C}_2(r)$. $\mathcal{C}_2(r)$ descends more rapidly initially, while $\mathcal{C}_1(r)$ descends more slowly initially but finally converges to a lower loss, as shown in Figure 4. Let c be the crossing point.

SHA-based methods: they use a set of fixed fidelities $\{r\}$ for both $\mathcal{C}_1(r)$ and $\mathcal{C}_2(r)$. If $r \le c$, then $\mathcal{C}_1(r) \ge \mathcal{C}_2(r)$, failing to indicate final performance.

FastBO: FastBO uses fidelities e_1 and e_2 for $\mathcal{C}_1(r)$, $\mathcal{C}_2(r)$. Clearly, $e_1 > e_2$, leading to $\mathcal{C}_2(e_1) < \mathcal{C}_2(e_2)$. In what follows, we discuss two cases.

Case 1: $e_1 \ge c$ (including $c \le e_2 < e_1$ and $e_2 < c \le e_1$): It follows that $\mathcal{C}_1(e_1) \le \mathcal{C}_2(e_1)$. Thus, we have $\mathcal{C}_1(e_1) \le \mathcal{C}_2(e_1) < \mathcal{C}_2(e_2)$. Then, $\mathcal{C}_1(e_1) < \mathcal{C}_2(e_2)$ holds true, aligning with the final performance.

Case 2: $e_2 < e_1 < c$: Based on Definition 1, $\mathcal{C}_1(e_1) - \mathcal{C}_1(2e_1) \approx \delta_1$, $\mathcal{C}_2(e_2) - \mathcal{C}_2(2e_2) \approx \delta_1$. Subtracting yields $\mathcal{C}_1(e_1) - \mathcal{C}_2(e_2) = \mathcal{C}_1(2e_1) - \mathcal{C}_2(2e_2) + \delta_1'$, where δ_1' is a small threshold around δ_1. As $2e_1 \ge c$ exists, it implies $\mathcal{C}_1(2e_1) < \mathcal{C}_2(2e_2)$ based on Case 1, so $\mathcal{C}_1(e_1) < \mathcal{C}_2(e_2)$.

Therefore, FastBO offers better fidelities that can more reliably indicate final fidelity performance, including scenarios even when $e_1, e_2 < c$. □

9. Illustration on Efficient Point and Saturation Point

In § 4.1, we provide formal definitions for the efficient point and saturation point. Here, we provide a more intuitive understanding of the concepts.

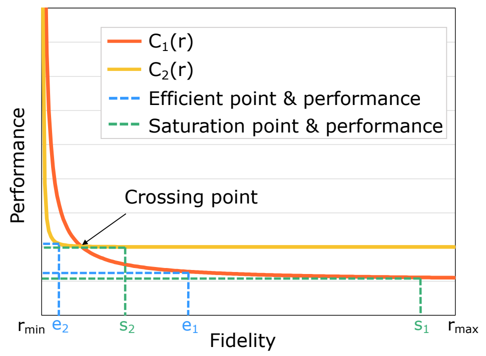


Figure 4. Illustration of efficient point and saturation point associated with learning curves.

Figure 4 shows an intuitive visualization of two learning curves $\mathcal{C}_1(r)$, $\mathcal{C}_2(r)$, together with their respective efficient points e_1, e_2 and saturation points s_1, s_2. We can easily grasp that the saturation points signify that the performance has nearly reached full convergence, while the efficient points, located at a relatively earlier stage, represent a position where performance can be achieved with high efficiency. From Figure 4, we can see a significant difference in the shapes of the two learning curves. $\mathcal{C}_2(r)$ experiences rapid initial descent and quick convergence; while $\mathcal{C}_1(r)$ experiences a slower initial descent, but eventually converges to a better performance than $\mathcal{C}_2(r)$. Due to this difference, we can find a crossing point where the two curves meet.

Suppose that $\mathcal{C}_1(r)$ and $\mathcal{C}_2(r)$ correspond to configurations $\boldsymbol{\lambda}_1$ and $\boldsymbol{\lambda}_2$ respectively, we can know $\boldsymbol{\lambda}_1$ outranks $\boldsymbol{\lambda}_2$ in terms of configuration performance ranking. Since FastBO utilizes efficient points as the fidelities for fitting the surrogate model, it is able to capture the distinctive trends in the learning curves. This ensures that the observed performance $y_1^{e_1}$ surpasses $y_2^{e_2}$, i.e., consistent with the configuration performance ranking. In contrast, existing successive halving-based methods may fail to maintain ranking consistency. Specifically, they are susceptible to erroneous termination of $\boldsymbol{\lambda}_1$ if the decision is made before the crossing point. Even with the aid of surrogate models, fitting before the crossing point leads to an inaccurate surrogate model.

Furthermore, we can observe that there is often a gap between the saturation point s_i and the final fidelity r_{max}, which becomes more pronounced on curves that converge rapidly, such as $\mathcal{C}_2$. FastBO utilizes the saturation point s_1 and s_2 as the approximation for the final fidelity r_{max}. Intuitively, $\boldsymbol{\lambda}_1$ and $\boldsymbol{\lambda}_2$ can achieve performances $y_1^{s_1}$ and $y_2^{s_2}$ that are very close to their performances at the final fidelity while saving a considerable amount of computational cost.

Table 3. The notations used throughout the paper and the corresponding definitions.

Notation	Definition
a	Acquisition function.
$c_j(r\|\boldsymbol{\theta}_j)$, $\mathcal{C}(r\|\boldsymbol{\phi})$	One of, and the combined parametric learning curve model.
$\mathcal{C}_i(r)$	Empirical learning curve for $\boldsymbol{\lambda}_i$.
$\mathcal{D}_i$	Observation set that used to fit the surrogate model, containing i pairs of data points.
e_i	The efficient point of $\boldsymbol{\lambda}_i$.
$f(\boldsymbol{\lambda})$, $f(\boldsymbol{\lambda}, r)$	Performance with configuration $\boldsymbol{\lambda}$ in the single-fidelity and multi-fidelity settings.
k	The number of configurations to be promoted.
$\mathcal{M}$	Surrogate model.
$\mathcal{O}_i^w$	Early observation set of $\boldsymbol{\lambda}_i$ across different fidelities, with a maximum level w considered.
r, r_{max}, r_{min}	Fidelity; the maximum and minimum fidelity.
s_i	The saturation point of $\boldsymbol{\lambda}_i$.
w	Warm-up point for all the configurations.
y_i, y_i^r	Evaluation results of $f(\boldsymbol{\lambda}_i)$ and $f(\boldsymbol{\lambda}_i, r)$ in the single-fidelity and multi-fidelity settings.
y_{max}, y_{min}	Best and worse possible evaluation performance.
α	Performance decrease ratio.
δ_1, δ_2	Small thresholds used in identifying efficient points and saturation points.
$\boldsymbol{\theta}_j$, $\boldsymbol{\phi}$	Parameters in one of, and the combined parametric learning curve model.
λ_i, $\boldsymbol{\lambda}$	A hyperparameter and a hyperparameter configuration.
Λ_i, $\boldsymbol{\Lambda}$	Domain of λ_i and search space of $\boldsymbol{\lambda}$.
ω_j	The weight of a parametric learning curve model.

10. Discussion on Choice of Parametric Learning Curve Models

In § 4.2, we construct the parametric learning curve model by combining three parametric models POW3, EXP3 and LOG2. Here, we provide detailed discussions on the choice.

Overall, POW3, EXP3 and LOG2, especially POW3, have shown good fitting and predicting performance in previous empirical studies [26, 43]. In order to capture the diversity in learning curve shapes, we explore different families of parametric models, including the power law, exponential, and logarithmic families. However, parametric models from the sigmoidal family, like MMF and Weibull, are not being considered, since they tend to fit well if enough observations are used for fitting; but in situations like ours where observations are limited, their performance is suboptimal [27]. Moreover, existing studies have discussed the underfitting of the power law and exponential models with two parameters and the overfitting of those with four or more parameters [20]. Therefore, we opt for the POW3 and EXP3 (i.e., power law and exponential models with 3 parameters respectively).

Considering the goal of high efficiency in HPO, we simplify the choice of the parametric learning curve model to strike a balance between capturing general learning curve shapes and prioritizing computational efficiency. We avoid considering complex models, since the computational complexity of the subsequent parameter estimation is proportional to the number of parameters. The increase in the number of parameters translates to an increase in the time required for each hyperparameter configuration during the optimization process, which runs counter to the fundamental objective of designing efficient HPO algorithms.

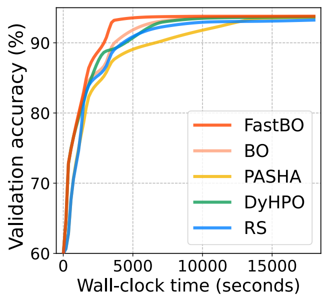


Figure 5. Performance of average validation accuracy on CIFAR-10 of the NAS-Bench-301 benchmark.

11. Extended Experiments

In this section, we provide additional experimental results and discussions.

11.1. Extended Experiments on NAS-Bench-301

Besides the comparison on the LCBench, NAS-Bench-201 and FCNet benchmark in § 5.1, we compare the anytime performance for the HPO methods on the NAS-Bench-301

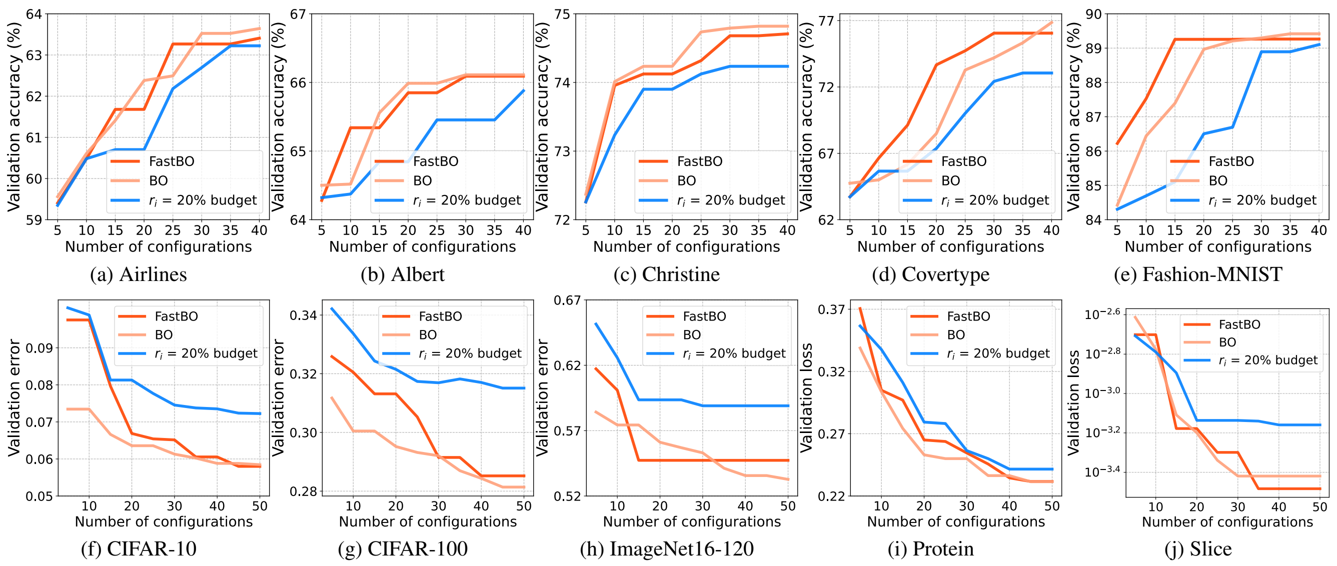


Figure 6. Performance of **(a)-(e)**: average validation accuracy against the number of evaluated configurations on the LCBench benchmark. **(f)-(h)**: average validation error against the number of evaluated configurations on the NAS-Bench-201 benchmark and **(i)-(j)**: average validation loss against the number of evaluated configurations on the FCNet benchmark.

benchmark [36] that has up to 10^{21} architectures on the DARTS/FBNet search space. The results on the CIFAR-10 dataset are shown in Figure 5. We can observe that FastBO still shows strong anytime performance on NAS-Bench-301, demonstrating the scalability of FastBO on large search spaces.

11.2. Extended Experiments of Sample Efficiency

In § 5.1, we show the anytime performance of a wide range of HPO methods. One reason for FastBO's good anytime performance is its good sample efficiency. Sample efficiency refers to the ability of an algorithm to find the optimal solution with the minimum number of samples. In the context of HPO, sample efficiency quantifies how effectively the algorithm explores the hyperparameter space and identifies promising configurations while minimizing the number of evaluated configurations. Methods with higher sample efficiency, such as BO, are capable of identifying satisfactory configurations with fewer evaluations.

To investigate the sample efficiency of FastBO, we conduct experiments using the same settings as the experiments in § 5.1 but plotting the achieved performance as a function of the number of evaluated configurations. Figure 6 shows the results obtained on the three benchmarks. We can observe that FastBO is able to achieve comparable, and in some cases, even superior performance to vanilla BO. It is particularly noteworthy considering that FastBO only performs partial evaluations of the configurations and is unsure about their performance at the final fidelity. The results demonstrate that FastBO has the ability to identify the appropriate fidelity for each configuration that can reliably indicate its performance. This ability is achieved by our adaptive strategy that adaptively finds the efficient point for each configuration as its fidelity r_i for surrogate model fitting.

In order to facilitate a clearer comparison, we also incorporate the results on an additional baseline: a partial evaluation scheme that replaces the adaptive strategy with the adoption of a fixed value as the fidelity for all the configurations to fit the surrogate model. We set the fixed fidelity to 20% of the total resource budget and present the results in Figure 6. We can see that this partial evaluation baseline consistently lags behind both FastBO and vanilla BO. It underscores the challenge of using a fixed fidelity value for all configurations in reflecting their final fidelity performance, which highlights the importance of the adoption of our adaptive strategy.

11.3. Extended Experiments of Anytime Performance

In § 5.1, we compare the anytime performance for the HPO methods. Here, we present the critical difference diagrams to summarize the ranks of all methods and provide information on the statistical difference.

Due to the potential inconsistencies in performance metric differences among different datasets within the same benchmark, which may affect the critical difference diagram, we first employ normalized regret to standardize each evaluation result y across datasets. The normalized regret for each y is defined as $(y - y_{min})/(y_{max} - y_{min})$, where y_{max} and y_{min} represents the best and worse possible eval-

Table 4. Configuration Evaluation Unit (CEU) of each dataset.

Benchmark	Dataset	CEU(second)
LCBench	Airlines	1187
	Albert	1297
	Christine	1715
	Covertype	1942
	Fashion-MNIST	831
NAS-Bench-201	CIFAR-10	3879
	CIFAR-100	3879
	ImageNet16-120	11150
FCNet	Protein	303
	Slice	547

uation performance can be found. Moreover, since different datasets require varying time to evaluate a single configuration, it is not fair or meaningful to use the evaluation results at a fixed time for all the datasets for comparison. Considering the varying dataset workloads, we introduce one Configuration Evaluation Unit (CEU) as the average time required to perform a complete evaluation of a single configuration on a given dataset. The CEU of each dataset shown in Table 4 is easy to obtain for the tabular benchmark.

With these ingredients, we provide the critical difference diagrams of LCBench, NAS-Bench-201 and FCNet in Figure 7. The critical difference diagrams are based on Wilcoxon-Holm post-hoc analysis. The results correspond to the results at one CEU[3], which represents relatively early evaluated performances. We can observe that FastBO consistently outperforms the baseline methods on all the benchmarks at one CEU, showing its capacity for an early advantage gain during the optimization process.

From Figure 7, we observe that the model-based multi-fidelity HPO methods, including FastBO, A-BOHB, A-CQR, BOHB, DyHPO and Hyper-Tune, outperform the other methods in most cases, highlighting the promising direction of integrating model-based approaches with multi-fidelity techniques. Among them, DyHPO also considers the learning curves of hyperparameter configurations. Both FastBO and DyHPO are able to gain an advantage at a relatively early stage, indicating the significant value of learning curve information in addressing HPO problems. However, we observe that DyHPO exhibits inferior performance on the FCNet benchmark, suggesting a potential limitation in dealing with the validation loss metric.

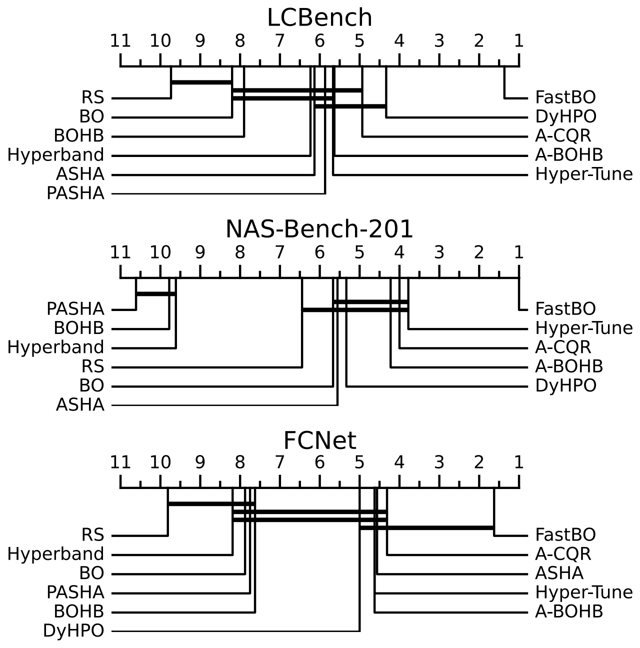


Figure 7. Critical difference diagram for LCBench, NAS-Bench-201 and FCNet at one CEU. The ranks indicate the sorted position in terms of normalized regret (the lower the better). Connected ranks indicate that differences are not statistically significant.

11.4. Extended Experiments of Efficiency on Configuration Identification

In § 5.2, we compare the time spent for the HPO methods on identifying a good configuration. Here, we report additional results on the datasets from the LCBench, NAS-Bench-201 and FCNet benchmarks in Table 5. We conduct experiments following the same settings as the experiments in § 5.2.

The experimental results shown in Table 5 are consistent with those shown in § 5.2. FastBO saves considerable wall-clock time over the baseline methods when achieving similar or better performance values, demonstrating the high efficiency of FastBO in identifying a good configuration. The model-free PASHA method often gets a high variance in wall-clock time because different random seeds can have a larger impact on it. Results of other model-free methods are not included in Table 5, since PASHA demonstrates its superiority over them [4].

11.5. Extended Experiments of Effectiveness of Adaptive Fidelity Identification

In § 5.3, we examine the effectiveness of the proposed adaptive fidelity identification strategy. Here, we provide additional results on more datasets.

We show the results on LCBench, NAS-Bench-201 and

[3]Note that the CEU is measured under one sequential worker, while FastBO and the baselines are evaluated under 4 parallel workers.

Table 5. Comparison of relative efficiency for configuration identification. Wall-clock time (abbr. WC time) reports the elapsed time spent for each method on finding configurations with similar performance metrics, i.e., validation error ($\times 10^{-2}$) and validation loss ($\times 10^{-2}$). Regarding relative efficiency, FastBO is set as the baseline with a relative efficiency of 1.00.

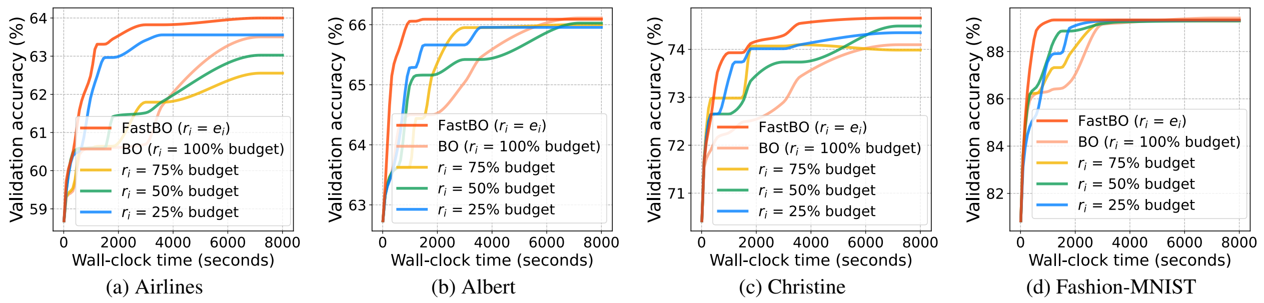
Dataset	Metric \ Method	**FastBO**	BO	PASHA	A-BOHB	A-CQR	BOHB	DyHPO	Hyper-Tune
Airlines	Val. error	**36.2$_{\pm 0.1}$**	36.3$_{\pm 0.5}$	**36.2$_{\pm 0.1}$**	36.3$_{\pm 0.3}$	38.9$_{\pm 0.5}$	38.5$_{\pm 0.1}$	36.3$_{\pm 0.1}$	**36.2$_{\pm 0.1}$**
	WC time (h)	**0.5$_{\pm 0.3}$**	2.4$_{\pm 1.3}$	1.1$_{\pm 0.7}$	1.1$_{\pm 0.6}$	2.7$_{\pm 0.6}$	2.2$_{\pm 0.4}$	1.3$_{\pm 0.3}$	1.1$_{\pm 0.6}$
	Rel. efficiency	**1.00**	0.23	0.51	0.50	0.20	0.25	0.38	0.48
Albert	Val. error	**33.9$_{\pm 0.1}$**	34.0$_{\pm 0.1}$	34.3$_{\pm 0.1}$	34.0$_{\pm 0.0}$	34.8$_{\pm 0.7}$	34.7$_{\pm 0.2}$	**33.9$_{\pm 0.2}$**	34.0$_{\pm 0.3}$
	WC time (h)	**0.5$_{\pm 0.3}$**	1.0$_{\pm 0.7}$	1.2$_{\pm 0.8}$	1.6$_{\pm 1.0}$	3.2$_{\pm 0.4}$	1.9$_{\pm 1.4}$	1.0$_{\pm 0.4}$	1.2$_{\pm 1.1}$
	Rel. efficiency	**1.00**	0.48	0.39	0.28	0.14	0.24	0.49	0.39
Christine	Val. error	**25.3$_{\pm 0.1}$**	25.5$_{\pm 0.1}$	25.6$_{\pm 0.1}$	25.5$_{\pm 0.1}$	26.7$_{\pm 0.0}$	26.8$_{\pm 0.2}$	25.5$_{\pm 0.1}$	25.4$_{\pm 0.0}$
	WC time (h)	**0.8$_{\pm 0.3}$**	2.4$_{\pm 1.3}$	2.4$_{\pm 2.2}$	2.1$_{\pm 1.2}$	1.6$_{\pm 2.1}$	1.5$_{\pm 0.9}$	1.6$_{\pm 0.6}$	2.9$_{\pm 0.8}$
	Rel. efficiency	**1.00**	0.33	0.33	0.37	0.48	0.54	0.47	0.27
Fashion-MNIST	Val. error	**10.7$_{\pm 0.1}$**	**10.7$_{\pm 0.1}$**	**10.7$_{\pm 0.1}$**	**10.7$_{\pm 0.1}$**	11.6$_{\pm 0.3}$	11.4$_{\pm 0.2}$	**10.7$_{\pm 0.1}$**	**10.7$_{\pm 0.1}$**
	WC time (h)	**0.2$_{\pm 0.1}$**	0.8$_{\pm 0.7}$	1.8$_{\pm 1.4}$	0.5$_{\pm 0.2}$	2.5$_{\pm 1.1}$	3.2$_{\pm 0.8}$	0.6$_{\pm 0.2}$	0.6$_{\pm 0.4}$
	Rel. efficiency	**1.00**	0.21	0.10	0.34	0.07	0.19	0.28	0.27
CIFAR-10	Val. error	**6.2$_{\pm 0.4}$**	6.5$_{\pm 0.4}$	6.4$_{\pm 0.7}$	**6.2$_{\pm 0.2}$**	6.3$_{\pm 0.4}$	6.3$_{\pm 0.2}$	6.3$_{\pm 0.4}$	**6.2$_{\pm 0.2}$**
	WC time (h)	**0.6$_{\pm 0.4}$**	3.9$_{\pm 2.0}$	1.3$_{\pm 0.6}$	2.3$_{\pm 1.1}$	2.6$_{\pm 0.9}$	2.1$_{\pm 0.5}$	2.5$_{\pm 0.8}$	1.6$_{\pm 0.8}$
	Rel. efficiency	**1.00**	0.16	0.49	0.27	0.25	0.31	0.26	0.39
CIFAR-100	Val. error	**28.7$_{\pm 1.3}$**	29.6$_{\pm 1.4}$	32.8$_{\pm 8.9}$	**28.7$_{\pm 1.2}$**	28.8$_{\pm 1.5}$	28.8$_{\pm 0.7}$	28.8$_{\pm 1.1}$	29.4$_{\pm 1.1}$
	WC time (h)	**1.2$_{\pm 0.9}$**	2.4$_{\pm 1.6}$	1.6$_{\pm 1.4}$	2.8$_{\pm 1.2}$	2.8$_{\pm 1.3}$	1.7$_{\pm 0.4}$	2.3$_{\pm 1.0}$	1.7$_{\pm 0.5}$
	Rel. efficiency	**1.00**	0.50	0.73	0.43	0.42	0.72	0.52	0.72
Protein	Val. loss	**22.6$_{\pm 0.4}$**	22.9$_{\pm 0.7}$	23.6$_{\pm 0.9}$	**22.6$_{\pm 0.3}$**	22.7$_{\pm 0.5}$	23.2$_{\pm 0.4}$	22.8$_{\pm 0.7}$	22.7$_{\pm 0.7}$
	WC time (h)	**0.3$_{\pm 0.1}$**	1.2$_{\pm 0.7}$	0.7$_{\pm 0.6}$	0.8$_{\pm 0.5}$	0.6$_{\pm 0.3}$	1.3$_{\pm 0.7}$	1.2$_{\pm 0.4}$	1.1$_{\pm 0.5}$
	Rel. efficiency	**1.00**	0.23	0.38	0.32	0.42	0.21	0.23	0.25

FCNet in Figure 8. FastBO with the adaptive fidelity identification strategy sets the efficient point e_i for each configuration $\boldsymbol{\lambda}_i$ as its fidelity r_i to fit the surrogate model. In contrast, the vanilla BO is a full evaluation scheme that uses 100% of the total resource budget as r_i. The other three baselines are also partial evaluation schemes like FastBO but they replace the adaptive choice of $r_i = e_i$ with a fixed fidelity, including 25%, 50%, and 75% of the total resource budget, for all the configurations to fit the surrogate model.

The results shown in Figures 8 are consistent with those shown in § 5.3. We have two main observations. Firstly, FastBO outperforms the other partial evaluation schemes that remove the adaptive fidelity identification strategy, showing the effectiveness of the proposed adaptive strategy. Secondly, although the partial evaluation schemes with fixed r_i are able to converge faster than the full evaluation counterpart (i.e., the vanilla BO) in the initial stage, this early advantage diminishes progressively over time. Finally, these partial evaluation baselines show significant differences in their final performance on 4 out of 7 datasets when compared to vanilla BO. The main reason is that these partial evaluation schemes naively use a fixed r_i for all the configurations and thus fail to create an accurate surrogate model to identify more promising configurations. This observation also highlights the importance of the adoption of our adaptive fidelity identification strategy.

11.6. Extended Experiments of Generality of The Proposed Extension Method

In § 5.4, we investigate the ability of our proposed extension method. Here, we provide additional results in Figure 9. We run three well-known single-fidelity methods CQR [34], BORE [42], and REA [32], and extend them to the multi-fidelity setting using our extension method, denoted as FastCQR, FastBORE, and FastREA respectively. More specifically, all the multi-fidelity variants evaluate the configurations to their efficient points and use the corresponding performances for the subsequent operations, i.e.,



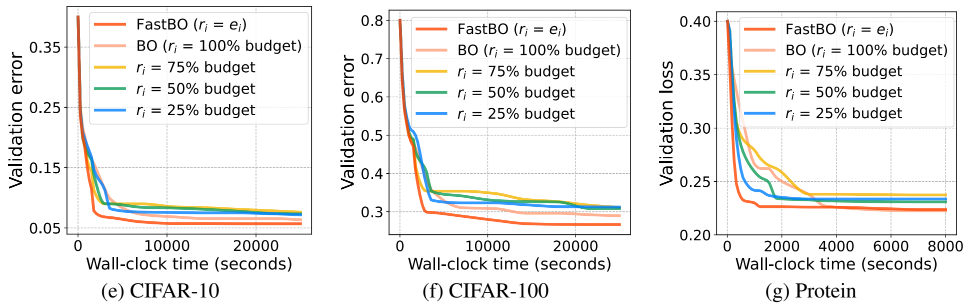
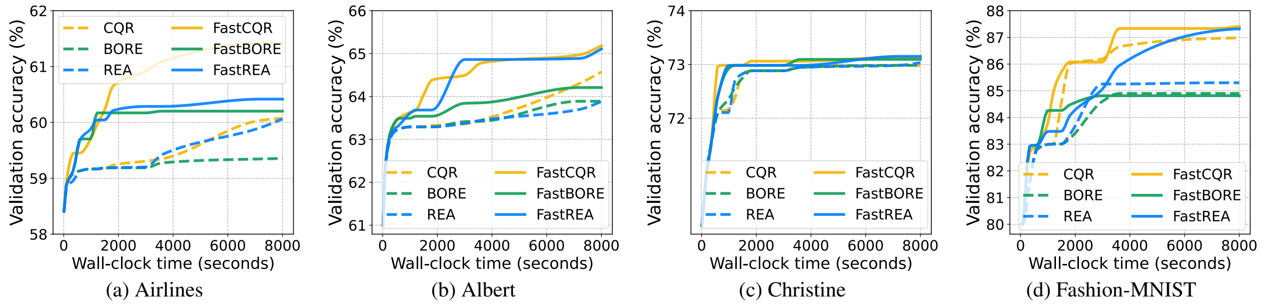


Figure 8. Average validation accuracy on the LCBench benchmark (**(a)-(d)**), average validation error on the NAS-Bench-201 benchmark (**(e)-(f)**), and average validation loss on the FCNet benchmark (**g**) of (i) FastBO that set $r_i = e_i$, (ii) the schemes that use fixed 25%, 50%, 75% of the total resource budget as r_i for all configurations, and (iii) vanilla BO that uses 100% total resource budget as r_i.



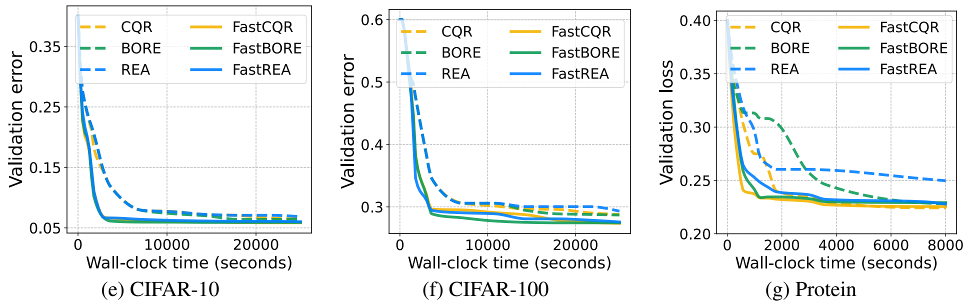


Figure 9. Performance of single-fidelity methods CQR, BORE, REA and their multi-fidelity variants FastCQR, FastBORE, FastREA using our extension method: average validation accuracy on the LCBench benchmark (**(a)-(d)**), average validation error on the NAS-Bench-201 benchmark (**(e)-(f)**), and average validation loss on the FCNet benchmark (**(g)**).

fitting the surrogate model for FastCQR and FastBORE, selection and variation for FastREA.

From Figures 9, we can clearly observe that the multi-fidelity variants with our extension method always outperform their single-fidelity counterparts. For the relatively simple task presented by the "Christine" dataset, the distinctions are not as pronounced as they are in the case of other datasets. However, it is still evident that the multi-fidelity methods are able to converge towards a higher accuracy more rapidly. Moreover, the evolutionary algorithm REA

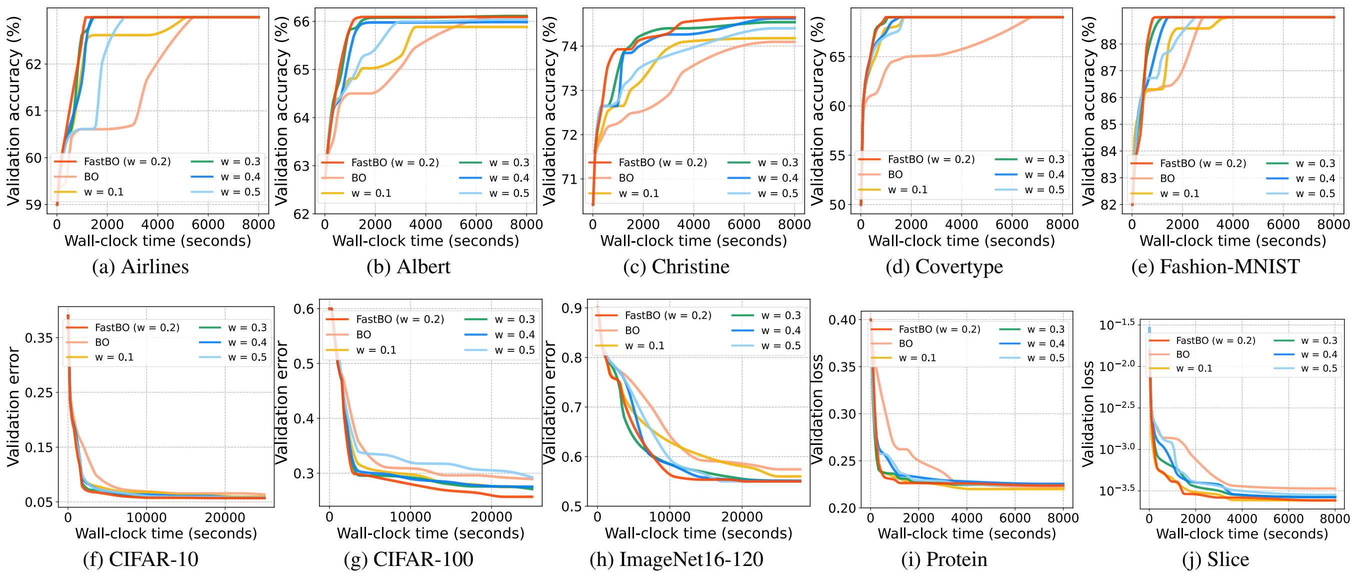


Figure 10. Performance of vanilla BO and the schemes with different w. The default setting of w in FastBO is 20% of the total resource budget (abbr. w = 0.2).

can also be enhanced by our extension method. The results are consistent with the observations shown in § 5.4 and highlight the wide applicability of the proposed adaptive strategy to extend any single-fidelity method to the multi-fidelity setting.

12. FastBO Hyperparameter Setting, Experiments, and Discussions

Here, we present the hyperparameter settings in FastBO, and provide experimental results and discussions on the hyperparameter settings.

12.1. Hyperparameter Setting

FastBO uses a Matérn $\frac{5}{2}$ kernel with automatic relevance determination parameters and the expected improvement acquisition function. We allocate 20% total resource budget for the warm-up stage, i.e., $w = r_{min} + 0.2 \cdot (r_{max} - r_{min})$. Ratio α is set to 0.1; thresholds δ_1 and δ_2 are set to 0.001 and 0.0005 [4]. We set k based on the number of parallel workers $\#workers$ and the number of started configurations $\#configurations$: $k = \max\{\lceil \#configurations/10 \rceil, \#workers\}$.

12.2. Experiments of Hyperparameter Setting

We compare the anytime performance of FastBO with different values of w, the warm-up point for all the configurations. We set w to 10%, 20%, 30%, 40%, and 50% of the total resource budget and examine their performances, where 20% one is the default setting of FastBO. In addition, we include a comparison with vanilla BO. In this section, we simply use $w = 0.1, ..., 0.5$ for abbreviation.

The results are shown in Figure 10. Overall, the default setting works quite well across different datasets. The results show that FastBO is not highly sensitive to the values of w, particularly within a reasonable range of 0.1 to 0.4, showing the robustness of our method.

Specifically, setting w to 0.2 and 0.3 always performs better on all the datasets. For $w = 0.5$, we can often observe a delayed performance improvement, as it requires more time to obtain additional evaluation observations for each configuration. Although this setting has the possibility of modeling more accurate learning curves, it wastes much time on expensive evaluations. The suitable values for w vary slightly across different benchmarks. For LCBench, the datasets have a relatively small maximum fidelity level of 50. Setting $w = 0.1$ cannot perform well, since there are only 5 observations for each configuration that can be used to fit its learning curve. While for NAS-Bench-201 and FCNet that have larger maximum fidelity levels, we can often see a delayed performance improvement when setting $w = 0.4$.

12.3. Discussion on Hyperparameter Setting

In order to avoid introducing extra efforts on tuning hyperparameters in FastBO, we intentionally set the hyperparameters in a simple way. We encourage the practitioners to directly use our default setting. Fine-tuning them is also a possibility and, if explored, may lead to further optimization on performance.

[4]Parameters δ_1 and δ_2 given here are derived after standardizing metrics to a uniform scale from 0 to 1.

Table 6. Detailed information of LCBench, NAS-Bench-201 and FCNet benchmarks.

Benchmark	#Evaluations	#Hyperparameters	#Fidelities
LCBench	2,000	7	50
NAS-Bench-201	15,625	6	200
FCNet	62,208	9	100

Table 7. Hyperparameters and configuration spaces for benchmarks.

Benchmark	Hyperparameter	Configuration space
LCBench	num_layers	[1, 5]
	max_units	[64, 512]
	batch_size	[16, 512]
	learning_rate	[1e-4, 1e-1]
	weight_decay	[1e-5, 0.1]
	momentum	[0.1, 0.99]
	max_dropout	[0.0, 1.0]
NAS-Bench-201	x0	[avg_pool_3x3, nor_conv_3x3, skip_connect, nor_conv_1x1, none]
	x1	[avg_pool_3x3, nor_conv_3x3, skip_connect, nor_conv_1x1, none]
	x2	[avg_pool_3x3, nor_conv_3x3, skip_connect, nor_conv_1x1, none]
	x4	[avg_pool_3x3, nor_conv_3x3, skip_connect, nor_conv_1x1, none]
	x3	[avg_pool_3x3, nor_conv_3x3, skip_connect, nor_conv_1x1, none]
	x5	[avg_pool_3x3, nor_conv_3x3, skip_connect, nor_conv_1x1, none]
FCNet	activation_1	[tanh, relu]
	activation_2	[tanh, relu]
	batch_size	[8, 16, 32, 64]
	dropout_1	[0.0, 0.3, 0.6]
	dropout_2	[0.0, 0.3, 0.6]
	init_lr	[0.0005, 0.001, 0.005, 0.01, 0.05, 0.1]
	lr_schedule	[cosine, const]
	n_units_1	[16, 32, 64, 128, 256, 512]
	n_units_2	[16, 32, 64, 128, 256, 512]

13. Experimental Setup

Here we provide more details on the experimental setup, including details of the used benchmarks and choice of parameters on the baseline methods.

13.1. Benchmark Details

In our experiments, we use 3 well-known tabular benchmarks: LCBench [48], NAS-Bench-201 [8], and FCNet [18]. We conclude detailed information on these benchmarks in Tables 6, including the number of provided evaluations, the number of hyperparameters, and the number of fidelities. Table 7 provides information on the hyperparameters in the benchmarks and their corresponding configuration spaces.

LCBench. LCBench is a neural network benchmark that consists of 2000 hyperparameter configurations. LCBench features a search space of 7 numerical hyperparameters of neural networks, including the number of layers, the maximum number of units per layer, batch size, learning rate, weight decay, momentum, and dropout. The fidelity refers to the number of epochs in LCBench and each hyperparameter configuration is trained for 50 epochs. LCBench contains 35 datasets and we run the 5 most expensive ones.

NAS-Bench-201. NAS-Bench-201 is a benchmark that consists of 15625 hyperparameter configurations. NAS-Bench-201 features a search space of 6 categorical hyperparameters that correspond to 6 operations within the macro architecture cell. The fidelity refers to the number of epochs in NAS-Bench-201 and each hyperparameter configuration, which represents a network architecture, is trained for 200 epochs. NAS-Bench-201 contains the image classification datasets cifar-10, cifar-100 and ImageNet16-120.

FCNet. FCNet is a benchmark that consists of 62208 hyperparameter configurations. FCNet features a search space of 4 architectural choices (i.e., the number of units and activation functions for two layers) and 5 hyperparameters (i.e., dropout rates per layer, batch size, initial learning rate and learning rate schedule). The fidelity refers to the number of epochs in FCNet and each hyperparameter configuration is trained for 100 epochs. FCNet uses 4 popular UCI datasets for regression and we run the 2 most expensive ones.

13.2. Choice of Parameters on Baseline Methods

We use implementations of all the baseline HPO methods provided in Syne Tune [33]. We here list the parameters used for running the baselines in our experiments. In general, we follow the default settings in Syne Tune which are also recommended in the previous work.

- Vanilla Bayesian Optimization (BO) [37] uses a Matérn $\frac{5}{2}$ kernel with automatic relevance determination parameters and the expected improvement (EI) acquisition function.
- ASHA [23], Hyperband [22] and PASHA [4] follow the successive halving (SHA) [16] framework and sample new configurations at random. We use the reduction factor η of 3 in all of them. In other words, the evaluations are stopped after 1, 3, 9, 27, ... resource levels.
- A-BOHB [19] follows the SHA framework with $\eta = 3$. It uses a stopping variant asynchronous scheduling, which is different from the promotion variant asynchronous scheduling used in ASHA. New configurations are selected as in the vanilla BO.
- A-CQR [34] follows the SHA framework with $\eta = 3$ and uses the promotion variant asynchronous scheduling as ASHA. It uses BO to select the configuration and uses the last observed values from the SHA framework to fit the surrogate model. It uses a conformal quantile regression-based surrogate model.
- BOHB [10] follows the SHA framework with $\eta = 3$ and uses synchronous scheduling. It uses BO with a multivariate kernel density estimator (KDE) to select new hyperparameter configurations.
- DyHPO [45] uses the introduced deep kernel Gaussian Process surrogate and multi-fidelity EI. It uses an RBF kernel and the dense layers of the transformation function have 128 and 256 units. It uses a convolutional layer with a kernel size of three and four filters.
- Hyper-Tune [24] follows the SHA framework with $\eta = 3$ and uses the promotion variant asynchronous scheduling as ASHA. It fits independent Gaussian process models at different fidelities.

The experiments in § 5.4 and Supplementary Material 11.6 contain three HPO methods and we use implementations of them provided in Syne Tune. We also provide the parameter settings of the three methods as follows.

- CQR [34] uses BO with a conformal quantile regression-based surrogate model to select new configurations.
- BORE [42] is evaluated with XGBoost [6] as the classifier with its default setting. We set $\gamma = 1/4$, consistent with BORE's default hyperparameter setting.
- REA [32] is an evolutionary algorithm that uses a population size of 10, and 5 samples are drawn to select a mutation from.