Towards Bandit-based Optimization for Automated Machine Learning

Amir Rezaei Balef, Claire Vernade and Katharina Eggensperger
Department of Computer Science, University of Tübingen
{amir.rezaei-balef, claire.vernade, katharina.eggensperger}@uni-tuebingen.de

Abstract

ML is essential for modern data-driven technology; however, its performance depends on choosing the optimal modelling approach, which can be prohibitive in low-resource settings. Efficient hyperparameter optimization (HPO) methods exist, but in practice, we often also need to decide between different model alternatives, e.g. deep learning or tree-based methods, giving rise to the combined algorithm and hyperparameter optimization problem (CASH). Automated Machine Learning (AutoML) systems address this problem typically by using HPO to search the joint hierarchical space of models and their hyperparameters. To increase efficiency, we study an alternative approach by conducting HPO for each model independently and trying to allocate more budget to the most promising HPO run. We use Multi-armed Bandits (MABs) as an efficient framework to balance exploration and exploitation in this setting and identify promising directions for future research. Concretely, we study how to leverage Extreme Bandits and propose two quantile-based algorithms to efficiently explore the extreme values for this practical setting. We empirically study the performance of state-of-the-art MAB methods on two AutoML benchmarks showing that even basic MAB methods with our adjusted regret term yield resource-efficient alternatives to searching the whole space jointly.

1 Introduction

The performance of machine learning (ML) solutions is highly sensitive to the choice of algorithms and their hyperparameter configurations. Finding a well-performing solution can be a daunting task for non-experts or under rigid resource constraints. AutoML aims to lower this barrier and to facilitate the application of ML. Hyperparameter optimization (HPO) methods focus on finding well-performing hyperparameter settings given a resource constraint, such as an iteration count or a time limit. Bayesian optimization (BO) (Jones et al., 1998; Snoek et al., 2012; Garnett, 2022) is a popular method for HPO and has been successfully demonstrated in practical settings (Falkner et al., 2018; Chen et al., 2018; Cowen-Rivers et al., 2022). BO is an iterative approach that fits a surrogate model and uses an acquisition function to find a promising configuration to evaluate next. However, in practice, it is often also unclear which ML model class would perform best on a given dataset. For example, on tabular data, an ubiquitous data modality in industry, it is heavily debated whether ensembles of gradient-boosted decision trees (Chen & Guestrin, 2016) or deep learning methods perform best (Gorishniy et al., 2021; Grinsztajn et al., 2022; McElfresh et al., 2023). Thus, we also need to choose the ML algorithm and run HPO to find its best-performing hyperparameters, giving rise to the combined algorithm selection and hyperparameter optimization problem (CASH) (Thornton et al., 2013). As a naive solution, we can run HPO independently for each algorithm and then compare the found solutions. Alternatively, AutoML systems typically address CASH by using a single instantiation of HPO to search a hierarchical space of models and hyperparameters (Thornton et al., 2013; Feurer et al., 2015). We illustrate this trade-off between searching the whole space and subspaces in Figure 1 (left), where we compare HPO for each model individually and on the whole space. Both approaches do not scale well with an increasing number of model classes

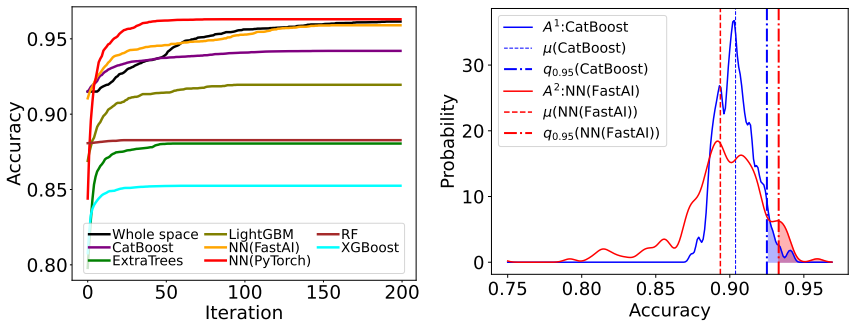

Figure 1: (Left) Accuracy over time for running HPO on each model class individually and HPO on the whole space (black line). (Right) The empirical performance distribution of two model classes on 200 sampled hyperparameter configurations.

and tight resource constraints. Ideally, we want a method that trades off exploring different model classes and exploiting HPO for promising model classes. Multi-Armed Bandits (MAB) have been developed for this setting but have received only little attention (Swearingen et al., 2017; Li et al., 2020; Hu et al., 2021). MABs are a promising direction to address the CASH problem, achieving high anytime performance for low-resource settings; thus, we revisit MABs for AutoML. Concretely, our contributions are as follows:

- By using a decomposed version of the CASH problem, we apply state-of-the-art methods of extreme bandits. To increase sample efficiency, we propose two MAB algorithms that are designed to find the arm with the highest τ-quantile.
- We empirically study performance on two AutoML benchmark tasks, and compare our MAB algorithms with popular MAB algorithms, prior MAB algorithms developed for AutoML, and HPO on the joint space of models and hyperparameters.
- We find that on the studied datasets, MAB algorithms are competitive compared to HPO on the joint space and that our MAB algorithms with a carefully designed regret term can yield superior performance (especially in the beginning).

2 Background on Aligning Multi-Armed Bandits for AutoML

A Multi-armed bandit (MAB) is a class of sequential optimization problems. The problem portrays a decision-maker that selects an action from a given set. After selection, they receive a reward generated by the unknown reward process of that arm. Under such uncertainty, at each round, the player may lose some reward due to selecting a sub-optimal arm. This loss is referred to as *regret*.

The definition of regret determines the strategy and the target of the optimization. MABs based on exploration-exploitation and pure exploration strategies are aligned with our setting. Most classical MAB problems aim to maximize the average rewards over time. However, in AutoML, we want to maximize the observed reward, which can be seen as selecting the arm with the heaviest tail. Figure 1 (right) illustrates this by showing the empirical performance distributions over the hyperparameter space of two ML models. Although CatBoost (blue) has a higher mean accuracy (and would be preferred by classical MAB methods), tuning the NN model (red) will eventually yield a higher performance (as seen by the higher value for the 95th percentile).

3 Problem Definition

Here, we study the CASH problem for supervised learning tasks, defined as follows (Thornton et al., 2013). Given a dataset $\mathbb{D} = \{D_{train}, D_{valid}\}$ of a supervised learning task, let $\mathcal{A} = \{A^{(1)}, ..., A^{(K)}\}$ be the set of K candidate ML algorithms, where each algorithm $A^{(i)}$ has its own hyperparameter search space $\mathbf{\Lambda}^{(i)}$. The goal is to search the joint algorithm and

hyperparameter configuration space to find the optimal algorithm A^* and its optimal hyperparameter configuration $\boldsymbol{\lambda}^*$ that minimizes loss metric $\mathcal{L}$, e.g., validation error. Formally,

$$A^*_{\boldsymbol{\lambda}^*} \in \underset{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}^{(k)}, A^{(k)} \in \mathcal{A}}{\arg\min} \mathcal{L}(A^{(k)}_{\boldsymbol{\lambda}}, \mathbb{D}), \tag{1}$$

For our approach, we decompose the problem into a bilevel optimization problem: at the lower level, we aim to find the best configuration $A^{(k)}_{\boldsymbol{\lambda}^*} \in \boldsymbol{\Lambda}^{(k)}$ for each model $A^{(k)}$, and at the upper level, we aim to find the optimal ML model $A^*_{\boldsymbol{\lambda}^*} \in \mathcal{A}$. Formally,

$$A^*_{\boldsymbol{\lambda}^*} \in \underset{A^{(k)} \in \mathcal{A}}{\arg\min}\, \mathcal{L}(A^{(k)}_{\boldsymbol{\lambda}^*}, \mathbb{D}) \qquad \text{s.t. } A^{(k)}_{\boldsymbol{\lambda}^*} \in \underset{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}^{(k)}}{\arg\min}\, \mathcal{L}(A^{(k)}_{\boldsymbol{\lambda}}, \mathbb{D}) \tag{2}$$

This decomposition has the following advantages: (1) Reduced complexity (by K small search spaces instead of one large) and (2) enabling efficient resource usage (by dynamically assigning resources to promising HPO runs). With this problem setup, we can get the best of both worlds: An online decision-making algorithm on the upper level to promote promising models and an HPO method on the lower level to find the best configuration efficiently.

4 Related Works

MABs have been studied before in the context of AutoML. Most notably, *Hyperband* (Li et al., 2018) uses successive halving (Jamieson & Talwalkar, 2016) to terminate evaluations, and (Ru et al., 2020) used MAB algorithms to optimize mixed hyperparameter spaces. Both methods are complementary to our setting as they would run on the lower level. *ATM* (Swearingen et al., 2017) is a scalable AutoML system that uses MAB algorithms for hyperpartition selection, where each hyperpartition is a set of hyperparameters for tuning.

Li et al. (2020) also decomposed CASH and proposed the *Rising Bandits* algorithm for the top level and SMAC (Hutter et al., 2011; Lindauer et al., 2022) as a Bayesian optimizer for the lower level. However, their algorithm is designed for increasing concave reward functions, which is a strong assumption. To weaken the assumption, they proposed "loose" concavity to describe the smooth growth rate of the reward sequences with a hyper-hyperparameter C. Their method can be sub-optimal for a small budget since a high value for C increases the initial exploration phase, and a small value for C would be unstable in many cases.

Similarly, the work by Hu et al. (2021) used random search or derivative-free optimization (Liu et al., 2017) at the lower level and proposed the *Extreme-Region Upper Confidence Bounds (ER-UCB)* algorithm for the upper level, which maximizes the extreme-region of feedback distribution. They compare two variants of the ER-UCB algorithm on 6 UCI classification datasets: ER-UCB-S for stationary feedback distribution with random search and ER-UCB-N for non-stationary distribution with a derivative-free optimizer. The proposed methods are sensitive to hyper-hyperparameters and assume that the rewards are Gaussian.

5 Methodology

Now, we turn to describing the MAB method that we use in the upper level of our problem setup, i.e. the first part of Equation 2. Given an overall budget of T iterations, at time step $t \leq T$, we define $\boldsymbol{\lambda}_t$ to be the configuration proposed by the optimizer in the lower level and $r_{i,t}$ to be the feedback to arm i obtained by evaluating $\boldsymbol{\lambda}_t$. To be consistent with the Bandit literature, we maximize the negative loss:

$$r_{i,t} = \max_{\boldsymbol{\lambda}_t \in \boldsymbol{\Lambda}^{(k)}} -\mathcal{L}(A^{(k)}_{\boldsymbol{\lambda}_t}, \mathbb{D}). \tag{3}$$

The goal is then to find the best-performing algorithm A^* and configuration $\boldsymbol{\lambda}^*$ in time $t \leq T$. We define regret $R(T)$ with I_t being the selected arm at time t by:

$$R(T) = \max_{k \leq K} \mathbb{E}[\max_{t \leq T} r_{k,t}] - \mathbb{E}[\max_{t \leq T} r_{I_t,t}]\,. \tag{4}$$

Equation 4 shows the definition of the regret for extreme Bandits (Baudry et al., 2022). Prior work found that state-of-the-art extreme bandits are not sample efficient enough in

practice for our problem formulation (Hu et al., 2021) due to the high variance and different types of extreme value distributions (Kotz & Nadarajah, 2000). Therefore, we reformulate the regret term to use quantiles instead of extreme values to provide a more robust understanding of data distributions with fewer samples. Furthermore, to operate under tight resource constraints, as typical in AutoML settings, we require strong anytime performance; therefore, we use cumulative regret $R_c(T)$ as below,

$$R_c(T) = \sum_{\mathcal{T}=1}^{T} \max_{k \leq K} \mathbb{E}[Q(r_{k,t}, \tau)]_{t \leq \mathcal{T}} - \mathbb{E}[Q(r_{I_t,t}, \tau)]_{t \leq \mathcal{T}} \tag{5}$$

with $Q(x, \tau) := \inf\{x : \mathbb{P}(X \leq x) \geq \tau\}$ computing the τ-quantile.

The distribution of sample quantiles, asymptotically, is a normal distribution as stated in Theorem 1 in Appendix A.1(Walker, 1968). However, it is not possible to theoretically compute the variance of this distribution in our setting since we do not have any assumptions on the probability density function f describing the reward distribution. We estimate the upper confidence bound of the quantile sequence; we call this algorithm *Quantile UCB* (blue highlighted text in Algorithm 1). In addition to calculating the empirical τ-quantile we also estimate the variance using conjugate analysis with normal data and pass it to *Bayes UCB* (Kaufmann et al., 2012) for Gaussian rewards with unknown variance, yielding *Quantile Bayes UCB* (red highlighted text in Algorithm 1).

We have two algorithms: *QuantileUCB*, a robust method for finding the best-performing configuration, and *Quantile Bayes UCB*, as a slightly more complex method with stronger expected anytime performance while being more sensitive to its prior hyperparameters (α_0, β_0).

Algorithm 1 Quantile UCB Quantile Bayes UCB

Require: τ (quantile parameter) α (exploration parameter) α_0, β_0 (prior parameters)

$t \leftarrow 1$
Pulls all arms once (observe rewards for default hyperparameter configurations)
for all arms $i \in K$ **do**
  Calculate the empirical τ-quantile $\hat{q}_i$ and its variance $\sigma(\hat{q}_i)$ for observed reward sequence $(r_{i,1}, ..., r_{i,n})$.
  Update $\alpha = \alpha_0 + \frac{n}{2}$ and $\beta = \beta_0 + \frac{n}{2}\sigma(\hat{q}_i)$ and calculate posterior mean $\hat{\sigma}_i = \frac{\beta}{\alpha - 1}$
  Update policy $U_i = \hat{q}_i + \sqrt{\frac{\alpha \log(t)}{n}}$ $U_i = Q(\mathcal{N}(\hat{q}_i, \hat{\sigma}_i), \tau = 1 - \frac{1}{t})$
end for
Select arm $I_t = \arg\max_{i \leq K} U_i$, observe reward $r_{I_t,t}$, normalize rewards, and $t \leftarrow t + 1$

6 Numerical Experiments

Next, we empirically compare our methods. We use the AutoML Toolkit (AMLTK)[1] framework for implementing pipelines and tasks, and SMAC (Lindauer et al., 2022) as a Bayesian optimizer at the lower level. For the upper level, we select several bandit algorithms as baseline methods, namely the classic UCB (UCB), extreme bandits (QoMax, QoSDA) (Baudry et al., 2022), Rising Bandits (Li et al., 2020) and ER-UCB-S (Hu et al., 2021). Additionally, we compare our two-level approach to using only SMAC or random search. We compute results on two AutoML tasks with 30 datasets each, *TabRepo* (Salinas & Erickson, 2023) a tabular benchmark [2] containing results for 200 randomly drawn configurations for 7 ML models and *YAHPO Gym* (Pfisterer et al., 2022), a surrogate benchmark with 6 ML models. We use a budget of 200 iterations and conduct 128 repetitions for each method and each AutoML task. We choose the hyperparameters for our MAB methods on a left-out set of datasets and provide details and results in Appendix A.2 and A.3.

We report averaged normalized loss for different time steps in Table 1 and average ranking in Figure 2 (and averaged normalized loss over time in Figure 3 in Appendix A.4). The result in Table 1 shows that tackling the decomposed CASH problem with any mentioned

[1] github.com/automl/amltk
[2] We restrict the HPO algorithms to only choose from the fixed set of runs in the table

MAB algorithm yields superior performance in the beginning ($T = 50$), being a powerful approach when we have a limited budget. Also, Quantile UCB and Quantile Bayes UCB outperform extreme bandits (QoMax, Max-Median) and classical MAB algorithms (UCB, Exp3), showing that using quantiles trades off sample efficiency and finding the arm with the highest possible rewards, overcoming the drawbacks of extreme bandits and classical MAB algorithms. The average rank shows a similar pattern: MAB methods yield better performance in the beginning, and in the long run, SMAC performs competitively on one of the two tasks. Comparing Quantile UCB and Quantile Bayesian UCB, we find that the Bayesian variant performs slightly superior but is also sensitive to its hyperparameters (see Appendix A.3), which we plan to investigate further in the future. Overall, we found that MAB methods are a competitive approach yielding strong initial performance.

Table 1: Average normalized error at steps ($T = 50, 100, 200$), lower is better.

Approach	TabRepo Normalized error			YAHPO Gym Normalized error		
	T=50	T=100	T=200	T=50	T=100	T=200
Quantile Bayes UCB	**0.35**	0.175	0.07	**0.298**	0.228	0.181
Quantile UCB	0.369	0.194	0.07	0.307	**0.227**	**0.17**
ER-UCB-S (Hu et al., 2021)	0.372	0.214	0.1	0.299	0.228	0.179
Rising Bandit (Li et al., 2020)	0.456	0.218	0.083	0.389	0.256	0.186
UCB	0.393	0.221	0.095	0.31	0.228	0.171
QoMax-SDA (Baudry et al., 2022)	0.441	0.225	0.089	0.364	0.245	0.185
Max-Median (Bhatt et al., 2022)	0.436	0.379	0.367	0.344	0.308	0.279
Exp3	0.458	0.275	0.13	0.378	0.279	0.206
SMAC (Lindauer et al., 2022)	0.479	**0.151**	**0.056**	0.422	0.233	0.2
Random Search	0.504	0.384	0.275	0.426	0.357	0.297

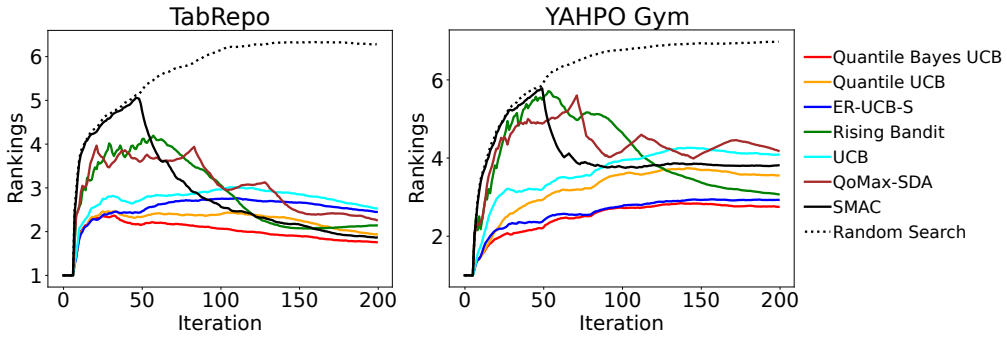

Figure 2: Average rank of each algorithm across 30 datasets, lower is better.

7 Conclusions and Future Work

In this paper, we studied MAB methods to trade off exploring different ML models and optimizing their hyperparameters. This is a core task in AutoML, aiming to democratize the usage of ML for non-experts and in low-resource settings. We explored Extreme Bandits and adjusted the regret term from extreme values to quantiles. We compared the resulting methods, Quantile UCB and Quantile Bayesian UCB, on two default AutoML tasks against state-of-the-art extreme bandits, prior approaches, and running HPO on the joint hierarchical space showing superior performance. We plan to conduct evaluations on a broader range of AutoML tasks , and in the future, extend this flexible framework with (a) multi-fidelity HPO (Falkner et al., 2018), (b) meta-learning across datasets and ML models, and (c) dynamically adding and eliminating arms (Jamieson & Talwalkar, 2016).

Acknowledgments

The authors are funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC number 2064/1 – Project number 390727645. Additionally, C. Vernade acknowledges funding from the DFG under the project 468806714 of the Emmy Noether Programme. The authors also thank the International Max Planck Research School for Intelligent Systems (IMPRS-IS).

References

D. Baudry, Y. Russac, and E. Kaufmann. Efficient algorithms for extreme bandits. *arXiv preprint arXiv:2203.10883*, 2022.

S. Bhatt, P. Li, and G. Samorodnitsky. Extreme bandits using robust statistics. *IEEE Transactions on Information Theory*, 69(3):1761–1776, 2022.

T. Chen and C. Guestrin. XGBoost: A scalable tree boosting system. In B. Krishnapuram, M. Shah, A. Smola, C. Aggarwal, D. Shen, and R. Rastogi (eds.), *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'16)*, pp. 785–794. ACM Press, 2016.

Y. Chen, A. Huang, Z. Wang, I. Antonoglou, J. Schrittwieser, D. Silver, and N. de Freitas. Bayesian optimization in alphago. *arXiv:1812.06855 [cs.LG]*, 2018.

A. Cowen-Rivers, W. Lyu, R. Tutunov, Z. Wang, A. Grosnit, R. Griffiths, A. Maraval, H. Jianye, J. Wang, J. Peters, and H. Ammar. HEBO: Pushing the limits of sample-efficient hyper-parameter optimisation. *Journal of Artificial Intelligence Research*, 74: 1269–1349, 2022.

S. Falkner, A. Klein, and F. Hutter. BOHB: Robust and efficient Hyperparameter Optimization at scale. In J. Dy and A. Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning (ICML'18)*, volume 80, pp. 1437–1446. Proceedings of Machine Learning Research, 2018.

M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter. Efficient and robust automated machine learning. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett (eds.), *Proceedings of the 29th International Conference on Advances in Neural Information Processing Systems (NeurIPS'15)*, pp. 2962–2970. Curran Associates, 2015.

R. Garnett. *Bayesian Optimization*. Cambridge University Press, 2022. Available for free at https://bayesoptbook.com/.

Y. Gorishniy, I. Rubachev, V. Khrulkov, and A. Babenko. Revisiting deep learning models for tabular data. In M. Ranzato, A. Beygelzimer, K. Nguyen, P. Liang, J. Vaughan, and Y. Dauphin (eds.), *Proceedings of the 35th International Conference on Advances in Neural Information Processing Systems (NeurIPS'21)*. Curran Associates, 2021.

L. Grinsztajn, E. Oyallon, and G. Varoquaux. Why do tree-based models still outperform deep learning on typical tabular data? In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, pp. 507–520, 2022.

Y. Hu, X. Liu, and S. Liand Y. Yu. Cascaded algorithm selection with extreme-region UCB bandit. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(10), 2021.

F. Hutter, H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In C. Coello (ed.), *Proceedings of the Fifth International Conference on Learning and Intelligent Optimization (LION'11)*, volume 6683 of *Lecture Notes in Computer Science*, pp. 507–523. Springer, 2011.

K. Jamieson and A. Talwalkar. Non-stochastic best arm identification and Hyperparameter Optimization. In A. Gretton and C. Robert (eds.), *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics (AISTATS'16)*, volume 51. Proceedings of Machine Learning Research, 2016.

D. Jones, M. Schonlau, and W. Welch. Efficient global optimization of expensive black box functions. *Journal of Global Optimization*, 13:455–492, 1998.

E. Kaufmann, O. Cappé, and A. Garivier. On bayesian upper confidence bounds for bandit problems. In *Artificial intelligence and statistics*, pp. 592–600. Proceedings of Machine Learning Research, 2012.

S. Kotz and S. Nadarajah. *Extreme value distributions: theory and applications*. world scientific, 2000.

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

Y. Li, J. Jiang, J. Gao, Y. Shao, C. Zhang, and B. Cui. Efficient automatic cash via rising bandits. In F. Rossi, V. Conitzer, and F. Sha (eds.), *Proceedings of the Thirty-Fourth Conference on Artificial Intelligence (AAAI'20)*. Association for the Advancement of Artificial Intelligence, AAAI Press, 2020.

M. Lindauer, K. Eggensperger, M. Feurer, A. Biedenkapp, D. Deng, C. Benjamins, T. Ruhkopf, R. Sass, and F. Hutter. SMAC3: A versatile bayesian optimization package for Hyperparameter Optimization. *Journal of Machine Learning Research*, 23(54):1–9, 2022.

Y. Liu, Y. Hu, H. Qian, C. Qian, and Y. Yu. Zoopt: Toolbox for derivative-free optimization. *arXiv preprint arXiv:1801.00329*, 2017.

D. McElfresh, S. Khandagale, J. Valverde, V. Prasad, B. Feuer, C. Hegde, G. Ramakrishnan, M. Goldblum, and C. White. When do neural nets outperform boosted trees on tabular data? In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 2023.

F. Pfisterer, J. van Rijn, P. Probst, A. Müller, and B. Bischl. Learning multiple defaults for machine learning algorithms. *arXiv:1811.09409v3 [stat.ML]*, 2018.

F. Pfisterer, L. Schneider, J. Moosbauer, M. Binder, and B. Bischl. YAHPO Gym – an efficient multi-objective multi-fidelity benchmark for hyperparameter optimization. In I. Guyon, M. Lindauer, M. van der Schaar, F. Hutter, and R. Garnett (eds.), *Proceedings of the First International Conference on Automated Machine Learning*. Proceedings of Machine Learning Research, 2022.

S. Pineda Arango, H. Jomaa, M. Wistuba, and J. Grabocka. HPO-B: A large-scale reproducible benchmark for black-box HPO based on OpenML. In J. Vanschoren and S. Yeung (eds.), *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*. Curran Associates, 2021.

B. Ru, A. Alvi, V. Nguyen, M. Osborne, and S. Roberts. Bayesian optimisation over multiple continuous and categorical inputs. In H. Daume III and A. Singh (eds.), *Proceedings of the 37th International Conference on Machine Learning (ICML'20)*, volume 98. Proceedings of Machine Learning Research, 2020.

D. Salinas and N. Erickson. Tabrepo: A large scale repository of tabular model evaluations and its automl applications. *arXiv preprint arXiv:2311.02971*, 2023.

J. Snoek, H. Larochelle, and R. Adams. Practical Bayesian optimization of machine learning algorithms. In P. Bartlett, F. Pereira, C. Burges, L. Bottou, and K. Weinberger (eds.), *Proceedings of the 26th International Conference on Advances in Neural Information Processing Systems (NeurIPS'12)*, pp. 2960–2968. Curran Associates, 2012.

T. Swearingen, W. Drevo, B. Cyphers, A. Cuesta-Infante, A. Ross, and K. Veeramachaneni. ATM: A distributed, collaborative, scalable system for automated machine learning. In *2017 IEEE International Conference on Big Data*, pp. 151–162. IEEE Computer Society, IEEE, 2017.

C. Thornton, F. Hutter, H. Hoos, and K. Leyton-Brown. Auto-WEKA: combined selection and Hyperparameter Optimization of classification algorithms. In I. Dhillon, Y. Koren, R. Ghani, T. Senator, P. Bradley, R. Parekh, J. He, R. Grossman, and R. Uthurusamy (eds.), *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'13)*, pp. 847–855. ACM Press, 2013.

A. Walker. A note on the asymptotic distribution of sample quantiles. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 30(3):570–575, 1968.

A Appendix

A.1 Theorem: Asymptotic normality of sample Quantiles

Theorem 1. *(Asymptotic normality of sample quantiles (Walker, 1968)) Let $(x_1, ..x_n)$ be IID draws from a CDF F with continuous density f. If $f(q_\tau) > 0$, we then have*

$$\mathcal{N}(q_\tau, \frac{\tau(1-\tau)}{f(q_\tau)^2}) \tag{6}$$

A.2 Experimental Setup

Here, we provide details on our experimental setups. We always use SMAC (Lindauer et al., 2022) as a Bayesian optimizer at the lower level, and for the upper level, we select several bandit algorithms as baseline methods, such as the classic UCB (UCB), extreme bandits (QoMax, QoSDA) (Baudry et al., 2022), Rising Bandits (Li et al., 2020) and ER-UCB-S (Hu et al., 2021). Additionally, we compare our two-level approach to only SMAC or random search. We run experiments for 200 iterations over 128 repetitions for each AutoML task. For a fair comparison, we always evaluate the default configuration for each model first and then allow SMAC to run an initial design of $50 - \#arms$ configurations in the upper level and $\frac{50}{\#arms-1}$ in the lower level.

As AutoML tasks, we considered default benchmark sets. First, we use data on 30 OpenML datasets from *TabRepo* (Salinas & Erickson, 2023) with 200 random configurations pre-evaluated for 7 baseline models. We set up SMAC to only work on a fixed set of configurations following prior work (Pineda Arango et al., 2021). Second, we use a subset of the *YAHPO Gym* (Pfisterer et al., 2018) containing surrogate HPO benchmark tasks for 30 OpenML tasks and 6 baseline models. We set the hyperparameters for our method to $\alpha = 0.25$ for *QuantileUCB* and $\alpha_0 = 1.0, \beta_0 = 0.2$ for *Quantile Bayes UCB* based on 18 additional datasets of *YAHPOGym* (see AppendixA.3 for more details) and use $\tau = 0.95$ as a quantile parameter for all algorithms. For the other MAB methods, we used the default setting.

A.3 Hyperparameter Settings

Table 2 shows the performance of SMAC (at $T \geq 100$) decreases when the number of initial designs is reduced to 0 configurations, and Table 3 shows results for different hyperparameter settings for our methods.

Approach	TabRepo Normalized error			YAHPO Gym Normalized error		
	T=50	T=100	T=200	T=50	T=100	T=200
SMAC	0.479	0.151	0.056	0.422	0.233	0.2
SMAC(no initial design)	0.361	0.187	0.074	0.362	0.301	0.243

Table 2: Average normalized error at steps ($T = 50, 100, 200$) for SMAC (with $50 - \#arms$ initial configurations) and SMAC (no initial design), lower is better.

Approach	Parameters	18 Tasks of YAHPO Gym Normalized Error T=50	T=100	T=200	Ranking
Quantile Bayes UCB	$\alpha_0 = 1.0, \beta_0 = 0.0$	0.381	0.333	0.31	2.031
Quantile Bayes UCB	$\alpha_0 = 1.0, \beta_0 = 0.01$	0.367	0.327	0.307	2.031
Quantile Bayes UCB	$\alpha_0 = 1.0, \beta_0 = 0.1$	0.306	0.247	0.216	1.729
Quantile Bayes UCB	$\alpha_0 = 1.0, \beta_0 = 0.2$	0.304	0.241	0.196	**1.639**
Quantile Bayes UCB	$\alpha_0 = 1.0, \beta_0 = 0.5$	0.334	0.244	0.177	1.67
Quantile Bayes UCB	$\alpha_0 = 1.0, \beta_0 = 1.0$	0.356	0.257	0.192	1.785
Quantile Bayes UCB	$\alpha_0 = 2.0, \beta_0 = 0.0$	0.381	0.333	0.31	2.031
Quantile Bayes UCB	$\alpha_0 = 2.0, \beta_0 = 0.01$	0.382	0.336	0.315	2.052
Quantile Bayes UCB	$\alpha_0 = 2.0, \beta_0 = 0.1$	0.35	0.31	0.274	1.913
Quantile Bayes UCB	$\alpha_0 = 2.0, \beta_0 = 0.2$	0.342	0.269	0.226	1.788
Quantile Bayes UCB	$\alpha_0 = 2.0, \beta_0 = 0.5$	0.311	0.251	0.196	1.698
Quantile Bayes UCB	$\alpha_0 = 2.0, \beta_0 = 1.0$	0.338	0.255	0.183	1.722
Quantile UCB	$\alpha = 0.1$	0.303	0.245	0.207	1.712
Quantile UCB	$\alpha = 0.25$	0.316	0.249	0.192	**1.667**
Quantile UCB	$\alpha = 0.5$	0.331	0.254	0.187	1.719
SMAC (Lindauer et al., 2022)		0.423	0.234	0.184	
Random Search		0.428	0.355	0.299	

Table 3: Results of Quantile Bayes UCB with different hyperparameter setting on 18 datasets of *YAHPOGym*. The ranking shows the average rank of the algorithm compared to SMAC and Random Search (lower is better). We boldface the optimal parameters used for the main experiments according to the ranking metric.

A.4 Further Experimental Results

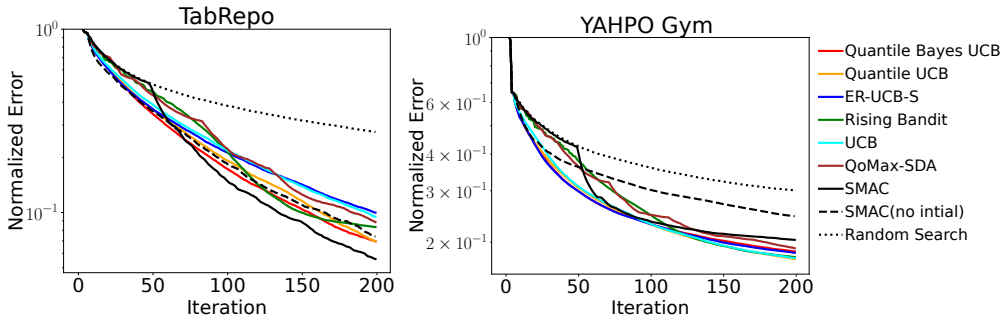

Figure 3: Average normalized error of each algorithm on the datasets, lower is better.