Model-Based Meta-Reinforcement Learning for Hyperparameter Optimization

Jeroen Albrechts Hugo Max Martin Maryam Tavakol

Abstract

Hyperparameter Optimization (HPO) plays a significant role in enhancing the performance of machine learning models. However, as the size and complexity of (deep) neural architectures continue to increase, conducting HPO has become very expensive in terms of time and computational resources. Existing methods that automate this process still demand numerous evaluations to find the optimal hyperparameter configurations. In this paper, we present a novel approach based on model-based reinforcement learning to effectively improve sample efficiency while minimizing resource consumption. We formulate the HPO task as a Markov decision process and develop a predictive dynamics model for efficient policy optimization. Additionally, we employ the `Deep Sets` framework to encode the state space, which is then leveraged in meta-learning for transfer of knowledge across multiple datasets, enabling the model to quickly adapt to new datasets. Empirical studies demonstrate that our approach outperforms alternative techniques on publicly available datasets in terms of sample efficiency and accuracy.

1 Introduction

The performance of machine learning models often relies heavily on the careful tuning of their hyperparameters. In the past, techniques based on random search were widely used for finding the optimal hyperparameter configurations and proved valuable for many years [2]. However, as models have grown in complexity and size, the performance of these methods has significantly declined. Consequently, the focus of research has shifted toward developing automated approaches that find the best hyperparameter settings within a constrained time or resource budget, which is known as Hyperparameter Optimization (HPO).

Prominent techniques for addressing the HPO problem are mainly based on Bayesian optimization [3, 29, 12], multi-armed bandits [4, 17, 25], or a combination of both [11]. Although these approaches show promise in dealing with blackbox functions under resource constraints, their computational cost and execution time rise significantly as the number of hyperparameters and/or their dimensionality increase, resulting in elevated sample complexity. Alternatively, Reinforcement Learning (RL) is a suitable choice to effectively tackle the HPO problem due to its ability to plan over a longer horizon, thus optimizing for an overarching goal instead of immediate benefits. In particular, model-based RL methods are highly sample-efficient by learning a model of the environment which can lead to an improved performance without needing as many real interactions compared to model-free counterparts [7]. Recently developed RL-based methods have been successful in improving sample efficiency for hyperparameter optimization [16, 19, 31, 20, 18]. Nevertheless, these techniques either discretize the hyperparameter space, decreasing the pool of solutions, or model the problem in ways that limit the agent's ability to plan over a longer horizon.

In this paper, we propose a model-based RL approach to address the HPO problem while facilitating knowledge transfer across diverse datasets via meta-learning. Inspired from Jomaa et al. [16] and Liu et al. [19], we model the problem as a Markov decision process (MDP) [28], where the search for the optimal set of hyperparameters unfolds as a sequential decision-making aimed at achieving a desired

17th European Workshop on Reinforcement Learning (EWRL 2024).

performance level or budget constraint. In this framework, the dynamics model is characterized using a predictive model to approximate the performance of different hyperparameter configurations, and the state space is embedded using the notion of `Deep Sets` [32]. We further incorporate a novel clipping technique and adapt the policy to maximize the best-found reward rather than the cumulative reward, to align with the objective of the HPO problem. Moreover, leveraging meta-learning with a certain similarity measure to compare various datasets enables both the predictive model and the policy to train with initial knowledge, accelerating the learning process and increasing sample efficiency. Meta-learning has been successfully applied to RL scenarios [9, 10, 21] and has proven effective in increasing convergence speed in the HPO context [31, 19, 20, 12].

Consequently, the main contributions of the paper are summarized as follows: (i) we introduce a novel model-based RL approach to tackle the HPO problem, (ii) we implement a meta-learning scheme to transfer model parameters across datasets, speeding up learning on new datasets, (iii) we leverage `Deep Sets` to learn an innovative state representation to consolidate the information of all previously tested hyperparameter configurations and their validation accuracy, (iv) we derive a modified objective function to enhance the performance of policy optimization, and (v) we evaluate the effectiveness of our method using six public datasets, where the experimental results showcase superior solution quality and sample efficiency of our approach compared to state-of-the-art techniques.

2 Related Work

Before automated hyperparameter selection, grid search [3] and random search [2] were common strategies for model tuning. However, grid search becomes impractical for large and continuous spaces, while random search lacks a learning component, leading to poor performance in complex tasks. Subsequently, Bayesian optimization techniques emerged to address the HPO problem by constructing surrogate Bayesian models from validation results [23, 29, 3]. Among these, tree-structured Parzen estimator (TPE) [3] enhances alternative methods by utilizing a tree-based model instead of Gaussian-like surrogates, allowing to handle various variable types. Still, TPE struggles in large-scale search spaces.

On the other hand, methods utilizing multi-armed bandits have shown promising performance in practical scenarios [4]. Hyperband is a well-known technique which iteratively narrows down the search space using the successive halving algorithm [17]. While this approach requires less computational resources and maintains a flexible balance between exploration and exploitation, it samples configurations only randomly, leading to reduced efficiency. HyperUCB [25] improves upon this by employing the upper confidence bound (UCB) algorithm [1] to leverage knowledge from previous evaluations. Alternatively, BOHB [11] combines the strengths of both Bayesian optimization and Hyperband, offering an efficient and scalable approach to optimal hyperparameter selection.

RL-based approaches to HPO aim to reduce the dimensionality of the search space by turning the search problem into a sequential decision-making task, selecting hyperparameter values one at a time rather than all at once [30, 18, 19, 31, 20]. Wu et al. [30] adapt this approach to generate one hyperparameter configuration per iteration, with validation loss computed at the end of the iteration, either directly or via a surrogate model. Liu et al. [18] improve this method by embedding the model into a meta-learning framework, leading to enhanced performance across different datasets. Other techniques incorporate meta-learning into policy optimization, utilizing additional context inputs [19], dataset representations [31], or experience variables [20] to guide decision-making. However, these augmentations often focus only on specific evaluated configurations, overlooking potentially useful information from other tested settings during training. Alternatively, Hyp-RL [16] presents a deep Q-network with a recurrent neural architecture for policy optimization, exploiting meta-features for pre-training on multiple datasets. Despite demonstrating promising results, this approach is limited to discrete hyperparameter values, and lacks permutation invariance.

3 Model-Based Meta-RL Framework

In this section, we present a model-based meta-Reinforcement Learning (meta-RL) approach to address the sample complexity in the HPO problem.

3.1 Preliminary

Let M denote a machine learning model with n hyperparameters to tune. The potential values for the i-th hyperparameter are determined within a domain Λ_i, leading to an overall hyperparameter space of $\Lambda = \Lambda_1 \times \Lambda_2 \times \cdots \times \Lambda_n$. Accordingly, M_λ represents the model M with a selected hyperparameter configuration $\lambda \in \Lambda$. The hyperparameter space consists of various value types, such as real values, integers, or categorical values. Given the dataset $\mathcal{D}$ which is divided into the training set $\mathcal{D}_{train}$ and validation set $\mathcal{D}_{valid}$, the optimal hyperparameter configuration λ^* can be obtained via

$$\lambda^* = \arg\min_{\lambda \in \Lambda} \mathcal{L}(M_\lambda | \mathcal{D}_{train}, \mathcal{D}_{valid}), \tag{1}$$

where $\mathcal{L}$ indicates the loss of model M on $\mathcal{D}_{valid}$ when trained using hyperparameters λ on $\mathcal{D}_{train}$, and Eq. 1 serves as the objective of the HPO problem.

3.2 MDP Formulation

In order to model HPO in an RL framework, we first characterize the problem as a Markov decision process (MDP) [28]. An MDP provides a mathematical framework for modeling sequential decision-making and is defined by a tuple $(\mathcal{S}, \mathcal{A}, T, r, \gamma)$, where $\mathcal{S}$ and $\mathcal{A}$ represent the state and action spaces, respectively, $T(s'|s, a)$ denotes the transition probabilities from state s to next state s' under an action a, $r(s, a)$ corresponds to the reward of taking a certain action a in a state s, and $\gamma \in (0, 1)$ is the discount factor. The objective of an MDP is to find an optimal policy $\pi^* : \mathcal{S} \to \mathcal{A}$ that maximizes the cumulative obtained rewards: $\pi^* = \arg\max_\pi \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right]$. To cast the HPO problem into an MDP, we first determine the action space $\mathcal{A}$ to be the set of all hyperparameter configurations Λ. Thus, at each timestep t, an action a_t corresponds to selecting a hyperparameter setting from this space, $a_t = \lambda_t \in \Lambda$. Subsequently, the reward is derived from the validation loss of the model on $\mathcal{D}_{valid}$, given configuration λ_t, after training the model on $\mathcal{D}_{train}$. Hence, the reward only depends on the current action and is expressed as $r(\lambda_t) = 1 - \mathcal{L}(M_{\lambda_t} | \mathcal{D}_{train}, \mathcal{D}_{valid})$, which we also denote as r for simplicity.

Furthermore, we construct the state space to exploit as much information from past evaluations as possible. Prior RL-based techniques either form hyperparameter configurations incrementally or incorporate the recent or optimal configurations in defining the state [18, 19, 20, 30, 31], which might overlook the information gained from other model evaluations. Inspired by Jomaa et al. [16], we include the set of all previous tested hyperparameter configurations and their rewards in defining the state at time t, $s_t = \{(\lambda_1, r_1), (\lambda_2, r_2), \ldots, (\lambda_{t-1}, r_{t-1})\}$. However, instead of using an RNN architecture for state encoding, which is *not* permutation-invariant, we employ `Deep Sets` [32] to learn a latent representation $\bar{s}_t$ of s_t independent of previous actions' order.

3.3 State Encoding

The importance of effective state encoding in our approach is twofold. First, the state space consists of sets of varying sizes at each time which requires an efficient embedding into a fixed-size latent space to be incorporated in the underlying RL algorithm. Second, adopting a unified representation for states will facilitate the meta-learning process across diverse datasets. Therefore, we employ `Deep Sets`, introduced by Zaheer et al. [32], to learn a robust state representation.

`Deep Sets` is a deep neural architecture designed to encode variable-sized data into fixed-size representations while guaranteeing permutation invariance, where its output remains the same regardless of the order of its input. We indicate this encoding procedure by the function f_ω, with parameters ω. In this framework, each set element is mapped into a latent vector using a shared encoder e_ω. Subsequently, the resulting vectors are merged through a "pooling" operator, such as *mean* or *sum*, combining all set items into a fixed-size vector. Finally, the output of the pooling operation feeds into a fully connected neural network d_ω, generating an output representation of constant length.

Consequentially, we employ this framework to learn the state representation in our problem, which is carried out as a standalone optimization process in the initial step of meta-learning. Given a model M to tune and d datasets $\{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_d\}$ for meta-training, we randomly sample m hyperparameter configurations $\{\lambda_1, \lambda_2, \ldots, \lambda_m\}$, and evaluate the validation loss of M_{λ_i} on each dataset $\mathcal{D}_j$, resulting in an action-reward pair of (λ_i, r_i^j). For each dataset $\mathcal{D}_j$, we further construct a set of states, where

Algorithm 1: Modified TD3 updates

input : Batch $\mathbb{B}_p = \{(\bar{s}_i, \lambda_i, r_i, \bar{s}'_i)\}$, max-reward r^*, ϕ, ϕ', θ, θ'

1 **for** $i \leftarrow 1$ **to** $|\mathbb{B}_p|$ **do**

2 $\quad \lambda'_i \leftarrow \text{clip}\big(\pi_{\phi'}(\bar{s}'_i) + \text{clip}(\epsilon, -[c, c]), [\alpha_{low}, \alpha_{high}]\big), \qquad \epsilon \sim \mathcal{N}(0, \sigma)$

3 $\quad y_i \leftarrow \max\left[r_i, \min[r^*, \min_{j=1,2} \gamma Q_{\theta'_j}(\bar{s}'_i, \lambda'_i)]\right]$

4 $\quad \delta_{i,j} \leftarrow Q_{\theta_j}(\bar{s}_i, \lambda_i) - y_i \qquad \forall j = 1, 2$

5 $\quad$ Update PER priorities using either $|\delta_{i,1}|$ or $|\delta_{i,2}|$ and compute w_i

6 Update critics by gradient descent using $\nabla_{\theta_j} \frac{1}{|\mathbb{B}_p|} \sum_i (w_i \cdot \delta_{i.j})^2 \qquad \forall j = 1, 2$

7 **if** *time for delayed update* **then**

8 $\quad$ Update actor by gradient ascent using $\nabla_\phi \frac{1}{|\mathbb{B}_p|} \sum_i Q_\theta(\bar{s}_i, \pi_\phi(\bar{s}_i))$

9 $\quad$ Update target parameters ϕ' and θ'

10 **return** $\phi, \phi', \theta, \theta'$

each state can comprise any subset of at least size one from $\{(\lambda_1, r_1^j), (\lambda_2, r_2^j), \ldots, (\lambda_m, r_m^j)\}$. Subsequently, for $l \in \{1, \ldots, 2^m - 1\}$, a state s_l^j is encoded into a latent representation via $\bar{s}_l^j = f_\omega(s_l^j)$. To train f_ω, we use the learned vectors in a classification task, where each state is categorized by the dataset to which it belongs, and the parameters ω are optimized with the cross-entropy loss.

3.4 Predictive Model

Model-based RL techniques learn an additional dynamics model, allowing for the generation of artificial rollouts for policy optimization, leading to enhanced sample efficiency over model-free methods [24]. In this section, we outline the design of our dynamics model, drawing inspiration from the MBPO algorithm [15].

The environment's dynamics consists of the transition function $T(s_{t+1}|s_t, a_t)$ and the reward r_t. T is deterministic in our scenario and can be derived from the actual action and reward, i.e., $s_{t+1} = \{s_t \cup (\lambda_t, r_t)\}$. Thus estimating r_t alone is sufficient to determine the one-step dynamics. Following the approach in [15], we employ an ensemble of k models characterized by parameters $\{\psi_1, \psi_2, \ldots, \psi_k\}$, each initialized differently with the aim of reducing the variance of predictions. Every model in the ensemble is then trained via maximum likelihood, $\psi_i \leftarrow \arg\max_\psi \mathbb{E}_{\mathbb{B}_0}[\log p_{\psi_i}(r_t|\bar{s}_t, \lambda_t)]$, where $p_{\psi_i}(r_t|\bar{s}_t, \lambda_t) = \mathcal{N}(\mu_{\psi_i}(\bar{s}_t, \lambda_t), \Sigma_{\psi_i}(\bar{s}_t, \lambda_t))$, and $\bar{s}_t = f_\omega(s_t)$ indicates the encoded fixed-size representation of s_t, as mentioned in the previous section.

The predictive model is optimized using the replay buffer $\mathbb{B}_0$ collected during meta-training procedure across all datasets (see section 3.7). However, this model may not generalize well to unseen datasets. To address this, we implement transfer learning by utilizing the (pre-)trained dynamics model and fine-tuning only its last few layers on the new dataset. Consequently, the predictive model can learn a more accurate reward with less data from the new dataset.

3.5 Policy Optimization

We employ Twin-Delayed Deep Deterministic policy gradient (TD3) [13] for policy optimization. TD3 is an actor-critic approach comprising a policy network π with parameters ϕ and two value networks Q for double Q-Learning, with parameters θ_1 and θ_2, collectively denoted as $\theta = \theta_1 \cup \theta_2$ for brevity. The parameters of the target networks are represented by ϕ', θ', respectively. The TD3 update procedure utilized in our HPO setting is outlined in Algorithm 1.

This algorithm is executed within the HPO procedure (see Section 3.6) on a batch $\mathbb{B}_p$ of transitions sampled via the predictive model as well as Prioritized Experience Replay (PER) [22]. Incorporating PER into TD3 enables the replay of important transitions, improving sample efficiency with slightly increased computation and memory costs. Therefore, we maintain priority values per each transition updated from TD errors (line 4-5). These values serve for both sampling the batch $\mathbb{B}_p$ from the replay memory and computing the importance-sampling weights w used in updating the critic networks (line 6). Furthermore, as in standard TD3, we update both critic networks in every iteration (line 6), while the policy and target networks are updated less frequently to ensure smoother updates (line 7-9). Additionally, exploration is managed by adding Gaussian noise with standard deviation σ, clipped

Algorithm 2: HPO procedure

input : Meta-parameters ϕ_0, θ_0, ψ_0, dataset $\mathcal{D}$ to tune, ω

1 $\phi, \phi', \theta, \theta', \psi \leftarrow \phi_0, \phi_0, \theta_0, \theta_0, \psi_0$
2 Reset λ^*, replay buffers $\mathbb{B}, \hat{\mathbb{B}}$, model errors $\mathcal{E}$, rewards $\mathcal{R}$
3 Initialize $\mathbb{B}$ with randomly sampled episodes using $\mathcal{D}$
4 **for** $e \leftarrow 1$ **to** E **do**
5 **if** *time to update the predictive model* **then**
6 $\psi_i \leftarrow \arg\max_\psi \mathbb{E}_{(\bar{s},\lambda,r)\sim\mathbb{B}}[\log p_{\psi_i}(r|\bar{s},\lambda)] \quad \forall i = 1,\ldots,k$
7 Reset $\mathcal{E}$
8 $s_1 \leftarrow \{(\lambda_0, r_0)\}$ with λ_0 randomly sampled, and r_0 its reward
9 $\bar{s}_1 \leftarrow f_\omega(s_1)$
10 **for** $t \leftarrow 1$ **to** T **do**
11 Select action $\lambda_t \leftarrow clip\big(\pi_\phi(\bar{s}_t) + \epsilon, [\alpha_{low}, \alpha_{high}]\big), \qquad \epsilon \sim \mathcal{N}(0, \sigma)$
12 Evaluate λ_t on $\mathcal{D}$ to obtain reward r_t
13 **if** $r_t > \max(\mathcal{R})$ **then** $\lambda^* \leftarrow \lambda_t$
14 $\bar{s}_{t+1} \leftarrow f_\omega(s_{t+1})$ where $s_{t+1} \leftarrow \{s_t \cup (\lambda_t, r_t)\}$
15 Append $(\bar{s}_t, \lambda_t, r_t, \bar{s}_{t+1})$ to $\mathbb{B}$ with max. priority
16 Append $|\hat{r}_t - r_t|$ to $\mathcal{E}$, with $\hat{r}_t \sim p_{\psi_i}(.|\bar{s}_t, \lambda_t)$, ψ_i is randomly selected
17 Append r_t to $\mathcal{R}$
18 **if** $\mathrm{mean}(\mathcal{E}) < \mathrm{std}(\mathcal{R})$ **then**
19 Generate a batch of data and add it to $\hat{\mathbb{B}}$
20 **if** *time to update the policy* **then**
21 Sample batch $\mathbb{B}_p \subset \mathbb{B}$ according to PER priorities
22 Sample batch $\hat{\mathbb{B}}_v \subset \hat{\mathbb{B}}$ s.t. $|\hat{\mathbb{B}}_v| = v|\mathbb{B}_p|$
23 $\mathbb{B}_p \leftarrow \mathbb{B}_p \cup \hat{\mathbb{B}}_v$
24 $\phi, \phi', \theta, \theta' \leftarrow$ **Algorithm1**$(\mathbb{B}_p, \max(\mathcal{R}), \phi, \phi', \theta, \theta')$ `# TD3 updates`
25 **return** λ^*

so its absolute value does not exceed a constant c, while clipping the next action to remain within a valid action range $[\alpha_{low}, \alpha_{high}]$ (line 2).

Moreover, to tailor TD3 to our HPO setting, we introduce two additional adaptations (line 3): (i) replacing the sum of rewards with the maximum reward and (ii) applying Q-value clipping. The first adjustment aligns with the HPO objective to identify a single hyperparameter setting that leads to the highest accuracy. This adjustment shifts the focus from maximizing the cumulative reward per episode to maximizing the highest obtained reward in an episode. Hence, inspired by [14], we use a *max* operation instead of *sum* when computing the target of the Bellman error y. The second adaptation involves a novel Q-value clipping method developed for our scenario. During meta-training, the parameters of the critic networks are initialized with meta-parameters which may not be close to zero and result in overestimation of learned Q-values. Higher Q-values reduce the effect of reward in target y, leading to low TD error δ and thus slow convergence. To address this, we cap the Q-values exceeding the maximum observed reward r^* recorded on the current dataset.

3.6 HPO Procedure

The HPO procedure outlined in Algorithm 2 illustrates the process of learning the optimal hyperparameter configuration λ^* for an unseen dataset $\mathcal{D}$ following the meta-training phase (see Section 3.7). For this procedure, the parameters of the dynamics model, policy, and value networks are initialized with the meta-trained parameters, denoted by ϕ_0, θ_0 and ψ_0, respectively (line 1).

This algorithm maintains two distinct replay buffers $\mathbb{B}$ and $\hat{\mathbb{B}}$. The former stores transitions from $\mathcal{D}$ and is initialized by randomly sampling hyperparameter settings and their validation loss using $\mathcal{D}$ (line 3), while the latter preserves transitions obtained from the learned dynamics model which generates artificial samples branching from actual data stored in $\mathbb{B}$. More details on the generation of samples can be found in Appendix A. We further ensure that $\hat{\mathbb{B}}$ contains admissible transitions by permitting the model to generate artificial data only when the *mean of its absolute errors*, logged in $\mathcal{E}$, is lower than the *standard deviation of the rewards* derived from the dataset, logged in $\mathcal{R}$, (line 16-18). This approach allows for automatic adjustment to the uncertainty in reward predictions and has shown promising empirical results.

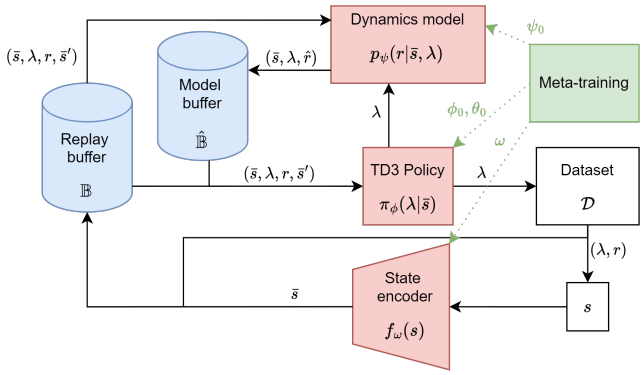


Figure 1: The overall RL-based HPO pipeline.

Furthermore, the algorithm iterates over E episodes each with a duration of T. Within every episode, s_t represents the state at timestep t, which consists of all evaluated hyperparameter configurations in the episode along with their corresponding rewards. The states are then encoded into the latent representation $\bar{s}_t$ and added to the replay buffer (line 14-15). While the first state is randomly sampled (line 8-9), subsequent steps in the episode follow the policy π_ϕ, employing TD3 clipping technique, and the true validation loss from the dataset (line 11-12). Additionally, model updates occur at specified intervals and when the length of the replay buffer $\mathbb{B}$ reaches a sufficient size. Consequently, all data in $\mathbb{B}$ is used to train the ensemble predictive model and $\mathcal{E}$ is cleared as the errors no longer accurately reflect the model's performance (line 5-7).

Moreover, periodically, we retrieve two mini-batches of data, one from the replay buffer $\mathbb{B}$ using PER priorities, and the other from the artificial data $\hat{\mathbb{B}}$, selected randomly but capped in size by a coefficient v. These batches are combined and leveraged to update the parameters of the actor and critic networks according to the modified TD3 update summarized in Algorithm 1 (line 20-24).

3.7 Meta Training

Meta-learning aims to acquire knowledge from prior tasks to facilitate learning of new tasks [26]. In the context of HPO, each dataset is treated as a distinct task, and the meta-training process operates on a set of datasets, called meta-datasets, enabling accelerated learning when applying the procedure to new datasets.

In our setting, meta-training involves iteratively applying the HPO procedure described in Section 3.6 to each meta-dataset, without resetting the parameters of the networks between iterations. Therefore, the initialization values for the actor (ϕ_0), critic (θ_0), and dynamics model (ψ_0) networks as well as the batch $\mathbb{B}_0$ are obtained from pre-trained models to aid learning in the subsequent iterations. The meta-training algorithm utilized in this phase closely resembles the HPO procedure in Algorithm 2, with three minor adjustments as outlined below. The complete algorithm is provided in Appendix B.

First, we refrain from sampling artificial transitions from the dynamics model, and thus discard the batch $\hat{\mathbb{B}}$, as time efficiency is not prioritized during the meta-training phase. Second, a meta-buffer $\mathbb{B}_0$ is maintained, storing all transitions $(\bar{s},\lambda,r,\bar{s}')$ across all meta-datasets, which serves for training the predictive model (see Section 3.4). The third modification addresses exploding action values by introducing action normalization [27]. During meta-training, carrying over parameters that lead to extreme hyperparameter values to a new dataset may result in limited exploration capabilities, as exploration relies on Gaussian noise centered on the policy. While this issue is less likely during testing in the HPO procedure, where the policy is trained briefly on a single dataset, it often leads to divergence and impedes policy learning during meta-training. Consequently, we perform action normalization to mitigate such undesirable behaviour. Given a hyperparameter configuration λ of dimension n, we compute the average absolute values of its components as $G = \frac{\|\lambda\|_1}{n}$. Accordingly, if $G > 1$, normalization is applied, resulting in a normalized action vector $\frac{\lambda}{G}$ for policy learning. The overall HPO pipeline is illustrated in Figure 1.

Table 1: Performance of different HPO methods with **fixed sample size** on 6 test datasets using both Random Forest (RF) and XGBoost (XGB).

		MBMRL-HPO		BOHB		TPE	
Dataset		Accuracy	std(%)	Accuracy	std(%)	Accuracy	std(%)
Teaching Assistant	RF	**0.696**	0.49	0.631	9.04	0.613	4.89
	XGB	**0.671**	0.44	0.657	1.48	0.660	0.95
Maternal Health Risk	RF	**0.851**	0.07	0.826	4.16	0.830	1.83
	XGB	**0.859**	0.06	0.858	0.19	**0.859**	0.22
Cervical Cancer	RF	**0.873**	0.11	0.871	0.00	0.871	0.00
	XGB	**0.875**	0.14	0.873	0.09	0.873	0.22
Obesity	RF	**0.966**	0.10	0.960	1.21	0.960	0.37
	XGB	0.969	0.12	0.973	0.04	**0.975**	0.13
Musk	RF	0.893	0.59	**0.898**	1.33	0.892	0.64
	XGB	**0.926**	0.30	0.918	0.14	0.919	0.26
Android Malware	RF	0.991	0.08	**0.992**	0.14	0.992	0.12
	XGB	**0.993**	0.00	0.993	0.05	0.993	0.05

4 Empirical Study

4.1 Experimental Setups

We carry out experiments to evaluate the performance of our approach, employing two popular classification techniques, namely Random Forest (RF) [5] and Extreme Gradient Boosting (XGBoost) [6], as the target blackbox models for the HPO procedure. In these experiments, we aim to tune five hyperparameters for RF and ten for XGBoost. We further select sixteen publicly available datasets [8] for the classification tasks, allocating ten sets for meta-training and reserving the remaining six sets for testing, with performance evaluation conducted via 10-fold cross-validation. Additional information about the datasets and the hyperparameter list is available in Appendices C and D. Moreover, we compare our approach, denoted as **MBMRL-HPO**, against two top-performing state-of-the-art methods, **TPE** [3] and **BOHB** [11], where both utilize Bayesian optimization, with the latter also being bandit-based.

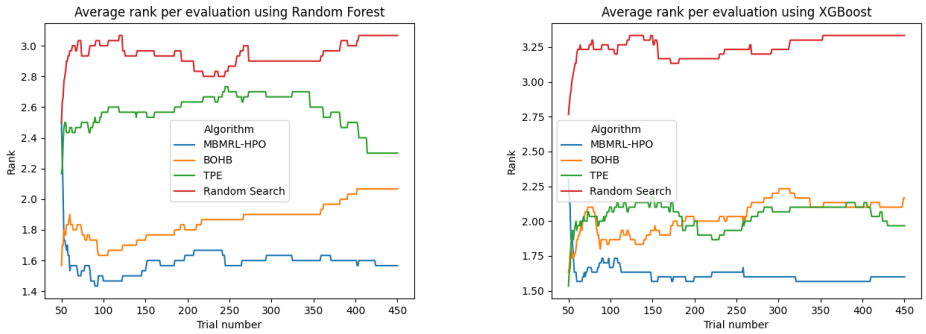
To measure the performance of HPO methods, we analyze both their computational efficiency and the quality of their optimal solution. The latter is determined by the accuracy of classification for a given hyperparameter configuration, while we impose a budget on the algorithm's computations in order to gauge the former. Initially, we set the budget based on the number of hyperparameter configurations to be evaluated, enabling us to assess sample efficiency in different methods (Section 4.2). Subsequently, in the remaining experiments, we enforce a time constraint, which is more in line with real-world situations to identify optimal hyperparameters within a limited timeframe (Section 4.3).

We select the HPO procedure's hyperparameters once and keep them fixed throughout all experiments for better generalizability across datasets and classification algorithms, refer to Appendix E for more details. An hyperparameter sensitivity analysis is performed in Section 5, while an ablation study can be found in Appendix F.

4.2 Performance Evaluation with Fixed Sample Size

We first limit the budget with the number of hyperparameter configurations to assess, set at a fixed total of 450 samples. For MBMRL, we randomly select the initial 50 evaluations, which are then employed to populate the buffer $\mathbb{B}$. Table 1 shows the performance of three HPO algorithms on the six test datasets in terms of accuracy of blackbox classification techniques. The values are averaged over five runs and include standard errors relative to the obtained results.

The results demonstrate that our approach consistently outperforms TPE and BOHB on most datasets for both classification algorithms. Specifically, it achieves superior performance in 9 out of the 12 experiments, and exhibits the lowest standard error in 8 experiments. These outcomes indicate that our algorithm not only requires fewer samples compared to its competitors but also offers greater reliability and consistency. Figure 2 further supports this finding in terms of average rank of different algorithms. In this context, the rank of a method on a dataset is determined by its relative performance



(a) Random Forest (b) XGBoost

Figure 2: Average ranking of four HPO methods over all runs and datasets for 450 iterations of hyperparameter evaluation using both Random Forest (a) and XGBoost (b). Lower ranks indicate better performance.

Table 2: Performance of HPO methods **under time limit** (5, 15, and 30 minutes) on 3 test datasets, averaged over 3 runs. K indicates the number of evaluations.

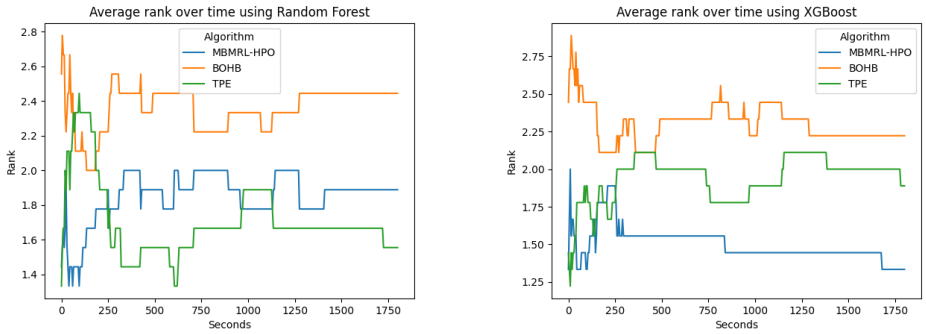
		MBMRL-HPO			BOHB			TPE		
Dataset–running time (min)		Accuracy	std(%)	K	Accuracy	std(%)	K	Accuracy	std(%)	K
Teaching Assistant–5	RF	**0.693**	1.36	619	0.6	6.85	1219	0.671	1.24	1447
	XGB	0.671	0.46	437	0.676	2.46	1022	**0.689**	0.91	1262
Teaching Assistant–15	RF	**0.698**	1.19	1657	0.622	5.82	3724	0.691	0.45	3992
	XGB	0.682	0.46	1223	0.682	1.66	3216	**0.702**	0.45	3325
Teaching Assistant–30	RF	**0.702**	0.45	3049	0.644	3.41	7378	0.698	0.45	7106
	XGB	0.691	0.45	2266	0.682	1.66	6398	**0.709**	0.44	5829
Musk–5	RF	0.887	0.19	255	**0.893**	1.02	301	0.892	0.48	684
	XGB	**0.924**	0.11	285	0.917	0.1	255	0.918	0.11	360
Musk–15	RF	**0.899**	1.19	653	0.894	0.89	976	0.897	0.67	1883
	XGB	**0.927**	0.32	773	0.920	0.22	848	0.920	0.18	1048
Musk–30	RF	0.901	1.06	1384	0.894	0.89	1965	**0.904**	0.57	3715
	XGB	**0.929**	0.18	1446	0.922	0.39	1725	0.922	0.11	2067
Android Malware–5	RF	**0.991**	0.05	87	0.987	0.41	64	**0.991**	0.16	110
	XGB	**0.993**	0.05	51	0.992	0.04	52	0.991	0.05	57
Android Malware–15	RF	0.991	0.05	283	0.991	0.3	246	**0.992**	0.05	339
	XGB	**0.993**	0.03	160	0.992	0.02	131	0.992	0.02	179
Android Malware–30	RF	0.991	0.05	610	0.991	0.3	526	**0.993**	0.08	680
	XGB	**0.993**	0.03	317	**0.993**	0.02	285	**0.993**	0.01	359

compared to other methods using the best hyperparameter configuration found on that dataset up to that point. Additionally, we include random search as a baseline in the plots, displaying notably inferior performance as expected. Further details on the results per each dataset are available in Appendix G.1.

4.3 Performance Evaluation under Time Limit

In the second series of experiments, we allocate a fixed amount of time for methods to execute and deliver their best-found hyperparameter configuration. Accordingly, we conduct the experiments across three test datasets, “Teaching Assistant", “Musk", and “Android Malware", each evaluated under 5-, 15-, and 30-minute time limits, resulting in a total of 9 setups per classification method. These datasets represent different scales of data size, with 150, 476, and 4464 datapoints respectively, leading to varying evaluation times. In each experiment, we record the highest performance achieved by each method at 5-second intervals, e.g., amounting to 360 measurements for a 30-minutes execution. Furthermore, we determine the total number of performed queries for evaluating the hyperparameter configurations in each experiment, represented by K.

Table 2 summarizes the performance of various HPO techniques based on the accuracy of classification models and the number of performed evaluations K, averaged over three runs with standard errors included. The results indicate that our approach achieves the highest performance in over half of the setups, while requiring fewer number of evaluations compared to BOHB and TPE. This performance improvement is particularly evident in the two smaller datasets, “Teaching Assistant” and “Musk”.



(a) Random Forest (b) XGBoost

Figure 3: Average ranks of HPO methods over all runs and datasets with 30-minute limit using Random Forest (a) and XGBoost (b). Lower ranks indicate better performance.

Table 3: Tested hyperparameter values. The column headed by MBMRL-HPO shows the values used in all other experiments.

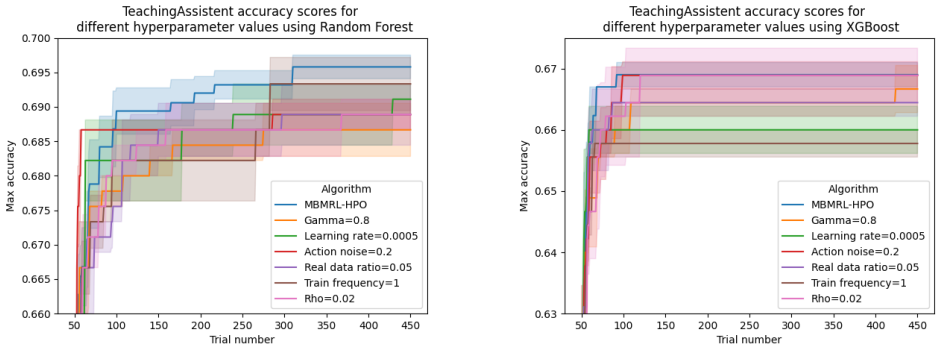
Parameter	MBMRL-HPO	Tested value
ρ	0.2	0.02
σ	0.1	0.2
γ	0.99	0.8
v	9	18
F	2	1
η	5e-5	5e-4

Despite TPE and BOHB evaluating more hyperparameter configurations in the same timeframe, MBMRL-HPO performs better or on par in most cases with fewer evaluations, which highlights its sample efficiency. This outcome also extends to the larger “Android Malware" dataset, where evaluations per hyperparameter configuration take longer, resulting in reduced K for TPE and BOHB. Nevertheless, the performance gap is less pronounced across all methods as classification results remain excellent for this dataset. Moreover, the increased runtime from 5 to 30 minutes yields less improvement in MBMRL-HPO compared to BOHB or TPE in all experiments, emphasizing the advantage of our approach within budgeted time constraints.

Additionally, Figure 3 displays the average ranks of three HPO algorithms during a 30-minute runtime. For XGBoost (b), our proposed approach outperforms both other methods, particularly after 300 seconds, clearly separating from TPE. On the other hand, MBMRL-HPO struggles to maintain its initial strong performance compared to TPE in Random Forest experiments (a). This is due to the fact that TPE can test more configurations in the same amount of time, especially with smaller datasets or faster underlying algorithms, which is the case for Random Forest. This pattern is also visible in Table 2, where generally all HPO methods are able to test more configurations for Random Forest than for XGBoost. Despite this, MBMRL-HPO shows superior performance in the early stages of the process, demonstrating the effectiveness of meta-learning within tight time constraints (less than 5 minutes). Besides, our approach excels in more complex tasks and larger datasets, such as tuning the computationally intensive XGBoost on the large dataset “Android Malware" where MBMRL-HPO obtains the best result. Detailed results for this experiment and other dataset-specific plots are provided in Appendix G.2.

5 Hyperparameter Sensitivity Analysis

In this section we briefly evaluate the influence of changing the values of certain hyperparameters. The changed hyperparameters with their tested values are displayed in Table 3, where ρ regulates the soft-update of the target networks, σ is the standard deviation of the noise added to actions in TD3, γ is the discount factor, v is the real data ratio used in TD3 updates, F is the number of hyperparameter configurations that elapse between gradient updates, and η is the learning rate of both actor and



(a) Random Forest (b) XGBoost

Figure 4: Performance of different hyperparameter settings of the MBMRL-HPO on the “Musk" dataset using both the Random Forest and XGBoost classification algorithms. The experiment was repeated 3 times, the hue indicates the standard errors.

critic models. In Figure 4, the influence of changing these hyperparameters can be observed on the “Teaching Assistant” dataset. For the Random Forest classification algorithm, the closest version in terms of performance to the proposed version is achieved when changing the training frequency from 2 to 1. While the final performance is only slightly worse, the result shows high deviation and low performance during the earlier stages of learning using this modification in combination with Random Forest. Increasing the action noise seems to slightly increase early performance, but sacrifices performance later on due to a lack of exploitation ability. However, we observe that all configurations achieve an average accuracy exceeding 0.680 relatively early in training, surpassing the final accuracies of 0.631 and 0.613 obtained with BOHB and TPE, respectively, on the same dataset (see Table 1).

When using XGBoost, changing hyperparameters does not seem to change the learning dynamics, as the performances increase during the first 100 trials and then remain constant for all configurations. Both increasing the action noise and decreasing ρ manages to achieve a similar final performance. However, they demonstrate slightly higher standard error. Changing the other hyperparameters lead to convergence to a lower performance for XGBoost, especially when increasing the learning rate or training frequency. This seems to be caused by overfitting on a small amount of data, not allowing for enough exploration to find a better solution. All configurations perform better than BOHB (0.657) on average, and a few configurations perform worse than TPE (0.660).

Overall, the performances with modified hyperparameters on this dataset are close to the original model’s performances, and remain for most of the configurations better than the performances achieved by the two other algorithms tested.

6 Conclusions

In this paper, we introduced an innovative model-based reinforcement learning approach to efficiently address the HPO problem. Our method employs meta-learning to accelerate the HPO process by transferring knowledge across various datasets. We further leveraged `Deep Sets` to learn a unified latent representation of the state space, and integrated reward maximization and Q-value clipping to enhance the performance of the TD3 algorithm. By conducting a comparative evaluation with state-of-the-art HPO techniques, we demonstrated the superior performance of our approach in several classification tasks, attributed to its high sample efficiency. Moreover, we confirmed the validity of our design choices through an hyperparameter sensitivity analysis and an ablation study.

References

[1] Peter Auer. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 3(Nov):397–422, 2002.

[2] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012.

[3] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, 24, 2011.

[4] Djallel Bouneffouf, Irina Rish, and Charu Aggarwal. Survey on applications of multi-armed and contextual bandits. In *2020 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE, 2020.

[5] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[6] Tianqi Chen, Tong He, Michael Benesty, Vadim Khotilovich, Yuan Tang, Hyunsu Cho, Kailong Chen, et al. Xgboost: extreme gradient boosting. *R package version 0.4-2*, 1(4):1–4, 2015.

[7] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *Advances in neural information processing systems*, 31, 2018.

[8] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

[9] Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, Ilya Sutskever, and Pieter Abbeel. RL^2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

[10] Rasool Fakoor, Pratik Chaudhari, Stefano Soatto, and Alexander J Smola. Meta-q-learning. *arXiv preprint arXiv:1910.00125*, 2019.

[11] Stefan Falkner, Aaron Klein, and Frank Hutter. Bohb: Robust and efficient hyperparameter optimization at scale. In *International Conference on Machine Learning*, pages 1437–1446. PMLR, 2018.

[12] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. *Advances in neural information processing systems*, 28, 2015.

[13] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.

[14] Sai Krishna Gottipati, Yashaswi Pathak, Rohan Nuttall, Raviteja Chunduru, Ahmed Touati, Sriram Ganapathi Subramanian, Matthew E Taylor, Sarath Chandar, et al. Maximum reward formulation in reinforcement learning. *arXiv preprint arXiv:2010.03744*, 2020.

[15] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. *Advances in Neural Information Processing Systems*, 32, 2019.

[16] Hadi S Jomaa, Josif Grabocka, and Lars Schmidt-Thieme. Hyp-rl: Hyperparameter optimization by reinforcement learning. *arXiv preprint arXiv:1906.11527*, 2019.

[17] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816, 2017.

[18] Xiyuan Liu, Jia Wu, and Senpeng Chen. Efficient hyperparameters optimization through model-based reinforcement learning and meta-learning. In *2020 IEEE 22nd International Conference on High Performance Computing and Communications; IEEE 18th International Conference on Smart City; IEEE 6th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 1036–1041, 2020.

[19] Xiyuan Liu, Jia Wu, and Senpeng Chen. A context-based meta-reinforcement learning approach to efficient hyperparameter optimization. *Neurocomputing*, 2022.

[20] Xiyuan Liu, Jia Wu, and Senpeng Chen. Efficient hyperparameters optimization through model-based reinforcement learning with experience exploiting and meta-learning. *Soft Computing*, 27 (13):8661–8678, Jul 2023. ISSN 1433-7479.

[21] Anusha Nagabandi, Ignasi Clavera, Simin Liu, Ronald S Fearing, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. *arXiv preprint arXiv:1803.11347*, 2018.

[22] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[23] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25, 2012.

[24] Richard S Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM Sigart Bulletin*, 2(4):160–163, 1991.

[25] Maryam Tavakol, Sebastian Mair, and Katharina Morik. Hyperucb: Hyperparameter optimization using contextual bandits. In *Machine Learning and Knowledge Discovery in Databases: International Workshops of ECML PKDD 2019, Würzburg, Germany, September 16–20, 2019, Proceedings, Part I*, pages 44–50. Springer, 2020.

[26] Sebastian Thrun and Lorien Pratt. Learning to learn: Introduction and overview. In *Learning to learn*, pages 3–17. Springer, 1998.

[27] Che Wang, Yanqiu Wu, Quan Vuong, and Keith Ross. Striving for simplicity and performance in off-policy drl: Output normalization and non-uniform sampling. In *International Conference on Machine Learning*, pages 10070–10080. PMLR, 2020.

[28] Chelsea C White III and Douglas J White. Markov decision processes. *European Journal of Operational Research*, 39(1):1–16, 1989.

[29] Jia Wu, Xiu-Yun Chen, Hao Zhang, Li-Dong Xiong, Hang Lei, and Si-Hao Deng. Hyperparameter optimization for machine learning models based on bayesian optimization. *Journal of Electronic Science and Technology*, 17(1):26–40, 2019. ISSN 1674-862X.

[30] Jia Wu, SenPeng Chen, and XiYuan Liu. Efficient hyperparameter optimization through model-based reinforcement learning. *Neurocomputing*, 409:381–393, 2020. ISSN 0925-2312.

[31] Jia Wu, Xiyuan Liu, and Senpeng Chen. Hyperparameter optimization through context-based meta-reinforcement learning with task-aware representation. *Knowledge-Based Systems*, 260: 110160, 2023. ISSN 0950-7051.

[32] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. *Advances in neural information processing systems*, 30, 2017.

A Data generation

The algorithm used to generate artificial data using the dynamics model can be found in Algorithm 3.

Algorithm 3: Dynamics model data generation
input : $\psi, \phi, \omega, \mathbb{B}$

Create empty batch $\hat{\mathbb{B}}$
for $i \leftarrow 1$ **to** H **do**
  $\bar{s}_i = f_\omega(s_i)$ where $s_i \sim \mathbb{B}$
  $\lambda_i \leftarrow clip\big(\pi_\phi(\bar{s}_i) + \epsilon, [\alpha_{low}, \alpha_{high}]\big), \quad \epsilon \sim \mathcal{N}(0, \sigma)$
  $\hat{r}_i \sim p_\psi(.|\bar{s}_i, \lambda_i)$
  $s'_i \leftarrow \{s_i \cup (\lambda_i, \hat{r}_i)\}$
  Append $(\bar{s}_i, \lambda_i, \hat{r}_i, f_\omega(s'_i))$ to $\hat{\mathbb{B}}$
return $\hat{\mathbb{B}}$

B Meta-Training Procedure

The meta-training algorithm can be found in Algorithm 4, along with an evaluation procedure in Algorithm 5 whose purpose is to only update the meta-parameters if the evaluation score improved over the course of the episode.

Algorithm 4: Meta-training
input : Set $\mathbb{T}$ of d datasets

Initialize ω and train f_ω on the datasets in $\mathbb{T}$, as described in Section 3.3
Initialize parameters ϕ_0, θ_0, evaluation score a^*, and buffer $\mathbb{B}_0$
for $\mathcal{D}_i \in \mathbb{T}$ **do**
  $\phi, \phi', \theta, \theta' \leftarrow \phi_0, \phi_0, \theta_0, \theta_0$ `# Initialize weights`
  Reset λ^*, rewards $\mathcal{R}$
  Fill $\mathbb{B}_i$ with randomly sampled episodes using $\mathcal{D}_i$
  for $e \leftarrow 1$ **to** E **do**
    $s_1 \leftarrow \{(\lambda_0, r_0)\}$ with λ_0 randomly sampled, and r_0 its reward
    $\bar{s}_1 \leftarrow f_\omega(s_1)$
    for $t \leftarrow 1$ **to** T **do**
      Select action $\lambda_t \leftarrow clip(\pi_\phi(\bar{s}_t) + \epsilon, \alpha_{low}, \alpha_{high}), \quad \epsilon \sim \mathcal{N}(0, \sigma)$
      $G \leftarrow ||\lambda_t||_1 / n$ `# Action normalization`
      if $G > 1$ **then** $\lambda_t = \lambda_t / G$
      Evaluate λ_t using $\mathcal{D}_i$ to obtain r_t
      $s_{t+1} \leftarrow \{s_t \cup (\lambda_t, r_t)\}$
      $\bar{s}_{t+1} \leftarrow f_\omega(s_{t+1})$
      Append $(\bar{s}_t, \lambda_t, r_t, \bar{s}_{t+1})$ to $\mathbb{B}_i$ with max. priority and to $\mathbb{B}_0$
      if *time to update the policy* **then**
        Sample batch $\mathbb{B} \subset \mathbb{B}_i$ according to PER priorities
        $\phi, \phi', \theta, \theta' \leftarrow$ **TD3_updates**$(\mathbb{B}, \max(\mathcal{R}), \phi, \phi', \theta, \theta')$
      if *end of episode* **then**
        `# Evaluate current episode performance`
        $a \leftarrow$ **Algorithm5**$(\mathcal{D}_i, \phi, \omega)$
        if $a > a^*$ **then**
          $a^*, \phi_0, \theta_0 \leftarrow a, \phi, \theta$
  Append $(\mathbb{B}_i, i)$ to $\mathbb{B}_0$
return ω, $\phi_0, \theta_0, \mathbb{B}_0$

Algorithm 5: Meta-training evaluation

input : $\mathcal{D}, \phi, \omega$

Initialize $\mathcal{R}$

$s_1 \leftarrow \{(\lambda_0, r_0)\}$ with λ_0 randomly sampled, and r_0 its reward

for $t \leftarrow 1$ **to** T **do**

  Select action $\lambda_t \leftarrow clip(\pi_\phi(f_\omega(s_t)), \alpha_{low}, \alpha_{high})$

  $G \leftarrow ||\lambda_t||_1 / m$ `# Action normalization`

  if $G > 1$ **then**

    $\lambda_t \leftarrow \lambda_t / G$

  Evaluate λ_t using $\mathcal{D}$ to obtain r_t, append r_t to $\mathcal{R}$

  $s_{t+1} \leftarrow \{s_t \cup (\lambda_t, r_t)\}$

$a \leftarrow \max(R) - \text{std}(R)$

return a

Table 4: Summary of dataset properties

Meta-training datasets			Testing datasets		
Name	Size	Dimension	Name	Size	Dimension
Predictive Maintenance	10000	6	Teaching Assistant	150	5
Heart Disease	296	13	Maternal Health Risk	1014	6
Tic Tac Toe	957	9	Cervical Cancer	668	30
Cylinder Bands	276	38	Obesity	2110	16
Car Evaluation	1727	6	Musk	476	166
Credit Approval	652	15	Android Malware	4464	241
Mice	552	77			
Climate Simulation	540	18			
BHP Flooding	1059	21			
Banknote Identification	1371	4			

C Datasets Characteristics

We detail the 16 datasets to evaluate the performance of the proposed algorithm on Table 4. These datasets are publicly available on the UCI database and are suitable for classification tasks. Due to limited computational resources, the size of the used datasets as well as the number of sets used is kept somewhat low. For simplicity, we removed datapoints containing missing values from the dataset. No further preprocessing is applied, since the goal of the experiments is solely to compare the algorithm's performance to that of existing methods, not to achieve the highest possible performance on the selected datasets.

D List of Hyperparameters

Table 5: Parameter names and bounds for the used classification algorithms

Random Forest		XGBoost	
Name	Range	Name	Range
n_estimators	[50,120]	max_depth	[3,15]
max_depth	[3,30]	eta	[0.001,0.1]
min_split	[2,100]	n_estimators	[50,200]
min_leaf	[1,100]	gamma	[0.05,1.0]
max_features	[0.1,0.9]	min_child_weight	[1,7]
		subsample	[0.6,1.0]
		colsample_bytree	[0.5,1.0]
		colsample_bylevel	[0.5,1.0]
		reg_alpha	[0,1.0]
		reg_lambda	[0.01,1.0]

The hyperparameters of RF and XGBoost that are tuned by the HPO methods are listed in Table 5.

Table 6: Hyperparameters used in MBMRL-HPO

F_{dyn}	10	Frequency of updates of the dynamics model
F	2	Frequency of TD3 updates
Q	10	Minimum length of buffer for dynamics model learning
k	7	Number of models in the ensemble
H	10000	Size of model generated batch
$\|\hat{\mathbb{B}}\|$	10000	Size of the artificial data buffer
v	9	Artificial data factor
η_m	0.001	Model learning rate
$\|\mathbb{B}\|$	50	Batch size
γ	0.99	Discount factor
ρ	0.2	Soft update hyperparameter for target networks
σ	0.1	Std of noise added to action for exploration
c	0.5	Clipping value of noise
α_{low}	-1	Minimum value allowed for the actions
α_{high}	1	Maximum value allowed for the actions
W	2	Frequency of updates of the actor and targets
η	$5e-5$	Learning rate of actor and critic

E Architectures and Hyperparameters of MBMRL-HPO

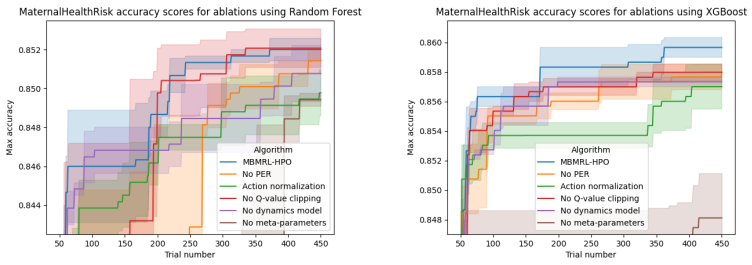
The architectures for the neural networks used are the following:

- **Encoder f_ω**: 3 hidden layers of 256 units for the shared encoder, *mean* pooling, 1 hidden layer of 256 units for the post-pooling network, *tanh* activation function
- **Dynamics model p_ψ**: 4 hidden layers of 200 units, *swish* activation function
- **Policy π_ϕ** and **Critic Q_θ**: 2 hidden layers of 256 units, *relu* activation function

The hyperparameters used for MBMRL-HPO can be found in Table 6.

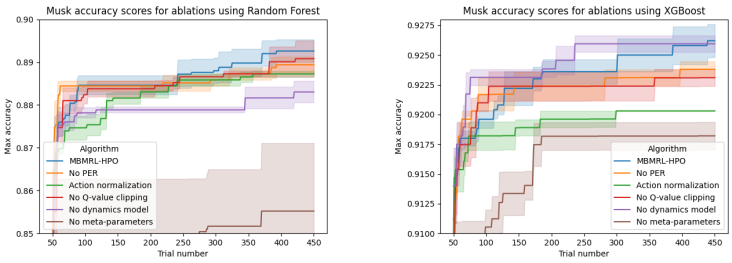
F Ablation Study

We perform an ablation study to analyze how various components of our approach contribute to its overall performance improvement. To this end, we compare the performance of the full algorithm denoted by MBMRL-HPO, with 5 configurations: (i) excluding PER, (ii) including action normalization during testing (in addition to meta-training), (iii) excluding Q-value clipping, (iv) excluding dynamics model predictions, and (v) excluding meta-training. Following this, the experiments are conducted with a fixed sample size budget of 450 evaluations on two datasets: “Maternal Health Risk" and “Musk". Figure 5 and 6 represent the accuracy of two classification methods when tuned with these instances of MBMRL-HPO, respectively for each dataset. The values are averaged across three runs, with hues indicating standard errors.



(a) Random Forest (b) XGBoost

Figure 5: Performance of different configurations of MBMRL-HPO in terms of accuracy of Random Forest (a) and XGBoost (b) on “Maternal Health Risk" dataset.



(a) Random Forest (b) XGBoost

Figure 6: Performance of different configurations of MBMRL-HPO in terms of accuracy of Random Forest (a) and XGBoost (b) on “Musk" dataset.

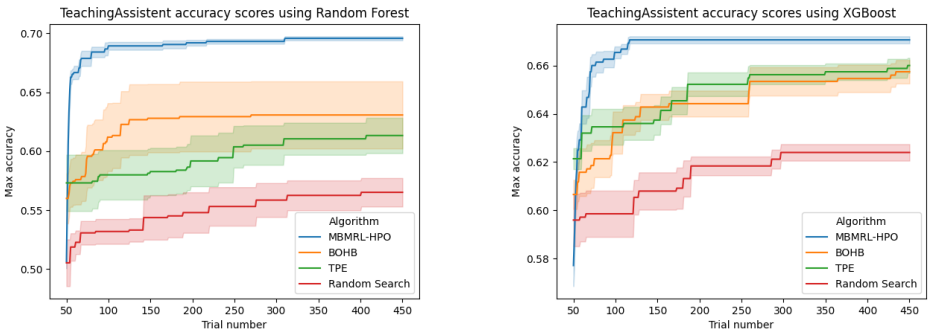
The plots demonstrate that MBMRL-HPO consistently outperforms its counterparts, with only two exceptions. Firstly, the configuration without Q-value clipping achieves comparable performance to MBMRL-HPO toward the end of Figure 5(a) when employing Random Forest on “Maternal Health Risk". However, its performance begins to improve only after 150 iterations, likely due to Q-value overestimation during the initial phase where clipping would have been beneficial. Furthermore, compared to MBMRL-HPO, it illustrates significantly inferior performance in Figure 6(b) and generally exhibits higher variance across all scenarios. Secondly, the instance without the dynamics model outperforms MBMRL-HPO when utilizing XGBoost on the “Musk" dataset in Figure 6(b), while it enhances the performance in all other experiments.

Overall, the results indicate that all modifications to the core algorithm positively impact the performance. The most advantageous addition is the incorporation of meta-learning, as the configuration lacking this feature (the brown curve) consistently performs worse in terms of both convergence speed and solution quality across all experiments. Additionally, we observe a decline in performance when employing action normalization during testing. This is likely caused by the narrowing exploration scope, potentially excluding areas of optimal solutions. Instead, action normalization proves more effective during meta-training to deal with exploding action values, as utilized in MBMRL-HPO.

G Additional Experiments

Here we provide a detailed breakdown of the results achieved on each dataset, which are only briefly summarized in the paper.

G.1 Performance Evaluation with Fixed Sample Size



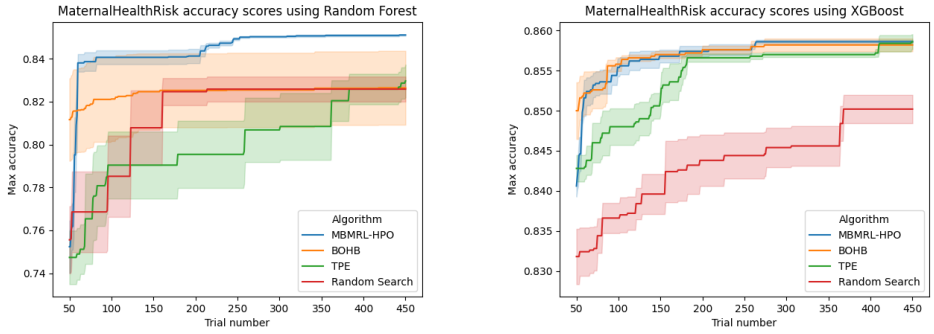
(a) Random Forest (b) XGBoost

Figure 7: Performance of HPO methods on “Teaching Assistant" dataset over 450 hyperparameter configurations using both the Random Forest and XGBoost classification algorithms. The experiment are repeated 5 times, and hues indicate standard errors.

Teaching Assistant

In Figure 7, the performance using the “Teaching Assistant" dataset can be observed. It can be noted that the proposed method, MBMRL-HPO, greatly outperforms the other algorithms in terms of final performance and sample efficiency for both classification algorithms. Namely, it manages to achieve fast convergence, surpassing the final performance of the other methods within less than 100 evaluations. Furthermore, the RL-based method achieves lower standard error, meaning that the results are more reliable.

Maternal Health Risk

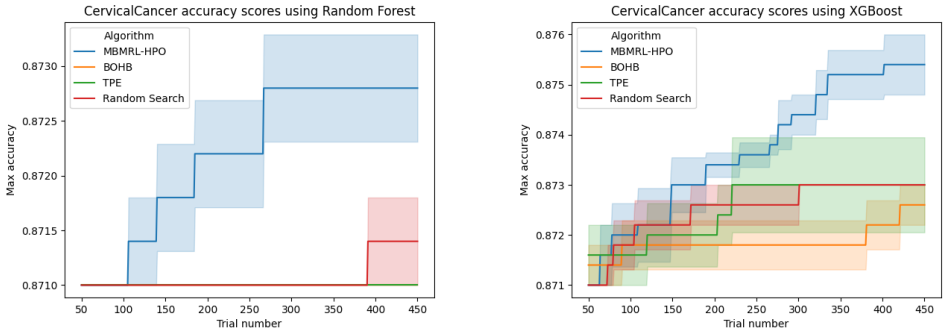


(a) Random Forest (b) XGBoost

Figure 8: Performance of HPO methods on “Maternal Health Risk" dataset over 450 hyperparameter configurations using both the Random Forest and XGBoost classification algorithms. The experiment was repeated 5 times, and hues indicate standard errors.

Our newly proposed HPO method also manages to achieve the highest final accuracy score using both classification algorithms on the “Maternal Health Risk" dataset. However, as can be seen in Figure 8, its sample efficiency using the XGBoost algorithm is closely matched by the BOHB algorithm. MBMRL-HPO manages to obtain a slight edge in the second half of the learning process and achieves lower variation. Using the Random Forest algorithm however, the performance of MBMRL-HPO is once again dominant. It finds a solution that significantly outperforms all other methods within a few evaluations and continues to improve this performance later in the process.

Cervical Cancer

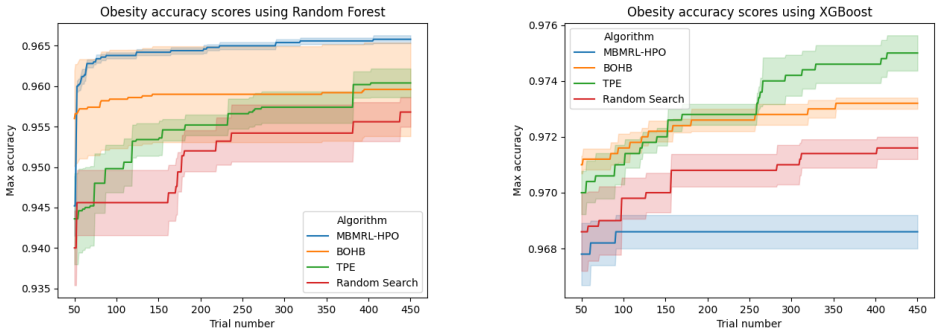


(a) Random Forest (b) XGBoost

Figure 9: Performance of HPO methods on “Cervical Cancer" dataset over 450 hyperparameter configurations using both the Random Forest and XGBoost classification algorithms. The experiment was repeated 5 times, and hues indicate standard errors.

For the “Cervical Cancer" dataset it is challenging for most methods to find a hyperparameter configuration that performs better than selecting one at random. Figure 9 shows that the only method able to continuously improve upon the best-found configuration is MBMRL-HPO. Although having a large deviation, it clearly outperforms other methods. Using XGBoost, all methods manage to achieve some improvement, but our proposed RL-based method once again performs best.

Obesity

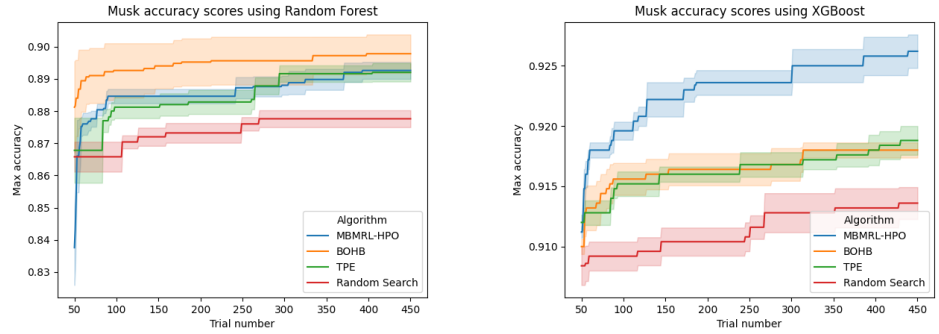


(a) Random Forest (b) XGBoost

Figure 10: Performance of HPO methods on “Obesity" dataset over 450 hyperparameter configurations using both the Random Forest and XGBoost classification algorithms. The experiment was repeated 5 times, and hues indicate standard errors.

MBMRL-HPO achieves mixed results on the “Obesity" dataset. As evident in Figure 10, it greatly outperforms all other methods in terms of convergence speed, final accuracy and reliability using the Random Forest algorithm. However, it achieves a significantly lower performance than all other methods using XGBoost, including the random search baseline. Upon further inspection of the results, this low performance seems to be the result of insufficient exploration. The algorithm gets stuck in a local minimum and is unable to find better solutions. This problem could likely be fixed by using a larger amount of initial random samples, but this would mean that the learning process starts later.

Musk



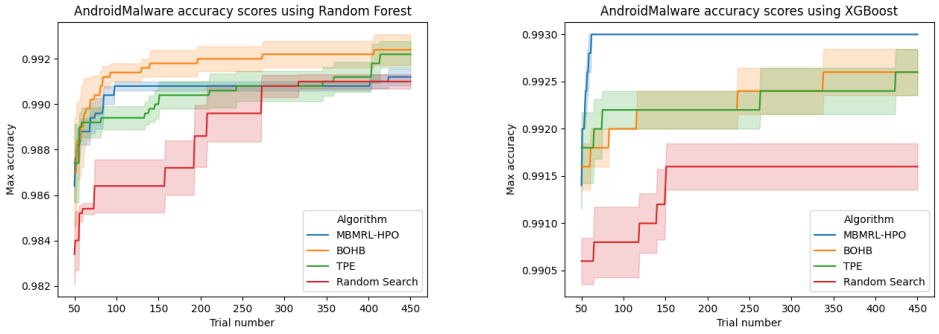
(a) Random Forest (b) XGBoost

Figure 11: Performance of HPO methods on “Musk" dataset over 450 hyperparameter configurations using both the Random Forest and XGBoost classification algorithms. The experiment was repeated 5 times, and hues indicate standard errors.

The performance of the RL-based method on the “Musk” dataset is also mixed. Figure 11 shows that using XGBoost, MBMRL-HPO easily outperforms the other methods. The performance of MBMRL-HPO using the Random Forest algorithm is however mediocre. The BOHB algorithm is

able to outperform MBMRL-HPO using Random Forest in terms of both convergence speed and final accuracy, but does so with greater variance.

Android Malware



(a) Random Forest (b) XGBoost

Figure 12: Performance of HPO methods on “Android Malware" dataset over 450 hyperparameter configurations using both the Random Forest and XGBoost classification algorithms. The experiment was repeated 5 times, and hues indicate standard errors.

The “Android Malware" dataset again shows variation in performance between the two used classification algorithms. Figure 12 shows that in the case of Random Forest, MBMRL-HPO performs mediocre, just slightly surpassing the random search method. Our method performs similarly to TPE, with TPE surpassing MBMRL-HPO’s final accuracy. BOHB outperforms both methods but also has a higher standard error. When using the XGBoost algorithm however, MBMRL-HPO converges rapidly and outperforms all methods by a large margin. It also manages to do so with little to no variance.

G.2 Performance Evaluation under Time Limit

In this subsection, we summarize the results of the time-based experiments and which are also illustrated using graphs. In these graphs, it can be seen how well the RL-based method performs using anywhere from 0 to 30 minutes of running time. The other used HPO algorithms are also plotted in the graphs, allowing for comparison with our proposed method.

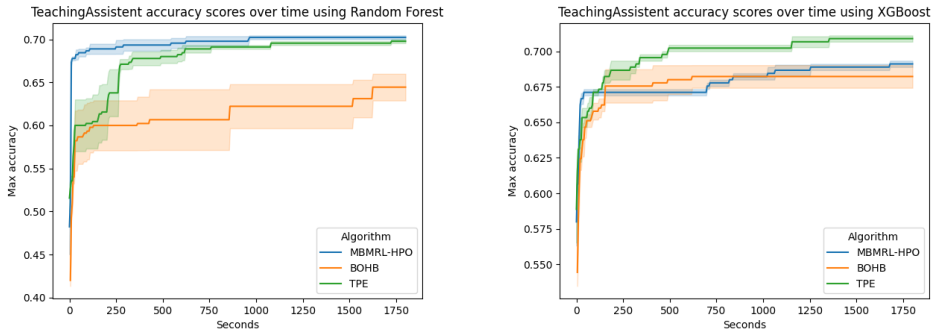
G.2.1 Teaching Assistant

Figure 13 shows that using Random Forest, MBMRL-HPO outperforms the other methods over the whole timeframe, despite performing a lot less evaluations as can be seen on Table 2 in the paper. It converges much more rapidly during the first minutes of training and keeps the lead until the end. The TPE method nearly catches up with MBMRL-HPO, while BOHB suffers from high variance and lower overall performance.

In contrary to Figure 7, MBMRL-HPO does not perform as well as other methods when using the XGBoost classification algorithm. Since the “Teaching Assistant" dataset is evaluated so rapidly, the first 450 evaluations are passed in less than a minute. It can be seen that similar to the experiments with fixed sample size, the RL-based method performs the best in these first few evaluations in the time-based experiment. It is however surpassed in performance quickly after that and is not able to catch up with the TPE method in terms of performance.

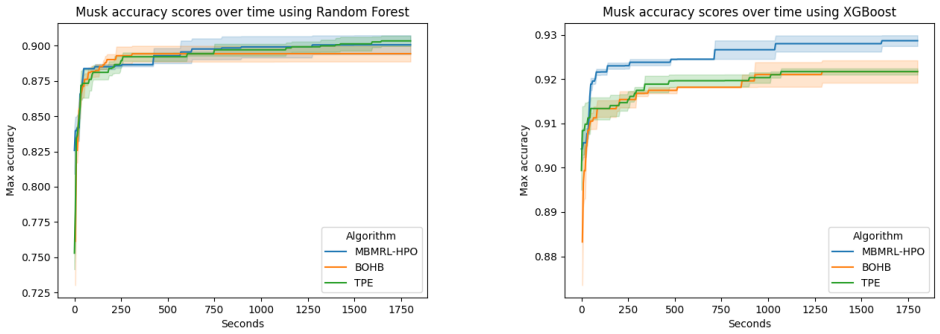
Musk

The results on the “Musk" dataset under time limit are again varying. This dataset is significantly larger than the “Teaching Assistant" dataset and therefore takes longer to evaluate. While the other two HPO methods execute evaluations quicker using this dataset, the difference is smaller in magnitude.



(a) Random Forest (b) XGBoost

Figure 13: Performance of HPO methods on “Teaching Assistant" dataset with 30 minutes running time using both the Random Forest and XGBoost classification algorithms. The experiment was repeated 3 times, and hues indicate standard errors.



(a) Random Forest (b) XGBoost

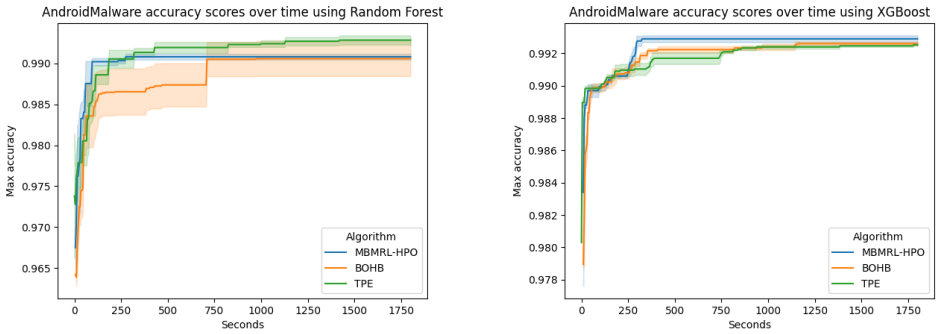
Figure 14: Performance of HPO methods on “Musk" dataset with 30 minutes running time using both the Random Forest and XGBoost classification algorithms. The experiment was repeated 3 times, and hues indicate standard errors.

For XGBoost, Figure 14 shows that MBMRL-HPO converges much more rapidly than the other methods. Furthermore, it continues to improve its performance over the whole experiment. In the case of Random Forest, the performance of MBMRL-HPO is comparable to the other methods. It achieves slightly better performance in the first few seconds, but comes in second in terms of accuracy behind the TPE algorithm after 30 minutes. It can however be noted that the upper bounds of the RL-based method’s average accuracy is higher than that of the other methods during the last three-quarters of the experiment, indicating that although having more variance, higher results can be obtained.

Android Malware

Compared to the other evaluated datasets, the “Android Malware" dataset is very large and contains many features. This makes the evaluation time of the classification algorithms relatively high. Table 2 of the paper shows that after 5 minutes, the MBMRL-HPO method has barely surpassed the 50 evaluations mark for both classification algorithms. Since the first 50 samples are selected at random, the learning process has only just started at this point in time. Furthermore, the number of evaluations at each point in time is very similar across all used HPO methods.

Figure 15 clearly shows the behaviour mentioned above when looking at subgraph (b). Around the 300 seconds or 5 minutes mark, the average maximum accuracy of MBMRL-HPO suddenly increases rapidly. This is caused by the fact that the 50 random samples are finished executing at this point and the RL agent starts selecting configurations according to the policy. While the performance of the



(a) Random Forest (b) XGBoost

Figure 15: Performance of HPO methods on “Android Malware" dataset with 30 minutes running time using both the Random Forest and XGBoost classification algorithms. The experiment was repeated 3 times, and hues indicate standard errors.

other methods keeps increasing as well, the superior sample efficiency of MBMRL-HPO results in faster convergence.

In the case of Random Forest, the performance of the newly proposed method is somewhat lacking. After the 50 random samples are executed after around 2 minutes of running time, the solution is not increased much. However, in this case the first 50 random samples used for exploration actually perform better than both the TPE and the BOHB methods. Therefore, increasing this amount might not affect the early performance drastically and could potentially increase the model’s final performance by increasing sample diversity allowing for more exploration. This effect could be analyzed in future research.