An Online Incremental Learning Approach for Configuring Multi-arm Bandits Algorithms

Mohammad Essa Alsomali[a,*], **Roberto Rodrigues-Filho**[b], **Leandro Soriano Marcolino**[a] **and Barry Porter**[a]

[a]School of Computing and Communications, Lancaster University, United Kingdom
[b]Department of Computing, Federal University of Santa Catarina, Brazil

Abstract. This paper introduces Dynamic Bayesian Optimisation for Multi-Arm Bandits (DBO-MAB), an algorithm that dynamically adapts hyperparameters of multi-arm bandit algorithms using incremental Bayesian optimisation. DBO-MAB addresses the challenge of tuning hyperparameters in uncertain and dynamic environments, particularly for applications like web server optimisation. It uses a dynamic range adjustment approach based on the interquartile mean (IQM) of observed rewards to focus the search space on promising regions. Evaluated across diverse static and dynamic environments, DBO-MAB outperforms state-of-the-art algorithms such as Bootstrapped UCB and f-Discounted-Sliding-Window Thompson Sampling, reducing average response time by $\approx 55\%$.

1 Introduction

The multi-armed bandit (MAB) model is a well-established paradigm for making decisions under uncertainty [15]. In this model, a set of available actions is represented as 'arms' that can be pulled. Each arm has an unknown reward distribution, so each repeated pull of the same arm provides an additional data point in the likely reward distribution of that arm. MAB models are appealing as control systems because their behavior has well-understood statistical bounds, and they have been used in optimisation problems for large-scale internet systems [6].

Numerous MAB algorithms are available, including Upper Confidence Bound (UCB1) [3], Sliding-Window UCB [11], Discounted UCB [11], and f-Discounted-Sliding-Window Thompson Sampling (f-dsw TS) [5]. However, applying these MAB algorithms in real-world systems presents challenges. For instance, UCB1 typically assumes rewards are normalized to the $[0, 1]$ or $[-1, 1]$ range. Factors like reward distribution, variance across arms, and differences between arm values make setting suitable hyperparameters for MAB algorithms (such as the window size, and discount factor) difficult when rewards may fall outside the usual ranges or in non-stationary environments. Converting real-world metrics like *milliseconds* or *kilowatts* into the $[0, 1]$ range is required for proper statistical behavior, but makes tuning difficult in the context of deployment environments for which we have real uncertainty, including uncertainty around likely reward distributions.

Manual tuning requires extensive system observation and is inefficient in changing environments, as new observations necessitate re-tuning. As Xie et al. [34] pointed out identifying the best MAB algorithm and tuning its hyperparameters is very time-consuming and heavily reliant on human expertise if done manually.

Existing hyperparameter tuning methods have limitations. For instance, the Population-Based Bandit (PB2) [28] algorithm optimizes scheduler hyperparameters and enables parallel execution. However, PB2 can be computationally demanding due to its reliance on a population of agents, each requiring significant resources. In the multi-armed bandit domain, Bootstrapping Upper Confidence Bound [14] presents a nonparametric, data-dependent UCB algorithm. It uses multiplier bootstrap to create tighter confidence bounds for better exploration and exploitation. However, it lacks dynamic adaptation of exploration constants based on system characteristics, making it less suited for deployment in uncertain conditions.

To address this limitation, firstly, we introduce a novel algorithm Dynamic Bayesian Optimisation Multi-arm Bandit (DBO-MAB). It is a tailored meta-learning algorithm for online and autonomous MAB hyperparameters adaptation based on Bayesian optimisation. Secondly, we enhances its performance through dynamic range adjustment using the interquartile mean (IQM) of observed rewards, a metric highlighted by Agarwal et al. [1] for its robustness, to focus the search space on promising regains. Our proposed method dynamically adjusts the hyperparameters of MAB algorithms at execution time, without the need for offline or manual tuning. The key ideas behind DBO-MAB are: (i) model the objective function using a Gaussian Process surrogate to represent uncertainty, (ii) employ Bayesian optimisation on the Gaussian Process to balance exploration and exploitation in hyperparameter selection, (iii) maintain fixed pillar points and the most recent points to capture changes as the hyperparameters are dynamically adjusted, (iv) dynamically adapt the MAB hyperparameters based on observed rewards, and (v) adjust the range for MAB hyperparameters dynamically.

The main contributions of this work are: (i) the novel DBO-MAB algorithm for online MAB hyperparameters adaptation, (ii) an extensive experimental evaluation highlighting the benefits of DBO-MAB, (iii) insights into preserving prior knowledge during online adaptation without restarting MAB algorithms each time a new parameter is tested (incremental learning process), and (iv) evaluation in dynamic environments where reward distribution changes over time.

Our experiments, conducted on a simulated web server composition optimisation problem, demonstrate that DBO-MAB reduces average response time by $\approx 55\%$ compared to Bootstrapped UCB.

* Corresponding Author. Email: {m.alsomali, l.marcolino, b.f.porter}@lancaster.ac.uk, roberto.filho@ufsc.br

2 Methodology

In this section, the UCB1 algorithm serves as a key example in our proposed approach. However, the underlying technique demonstrated is general and can be applied to other multi-armed bandit (MAB) algorithms as we will demonstrate later.

The UCB1 algorithm [3] is an online learning approach where the learning agent pulls arms (actions) in a sequential manner. The agent's goal is to maximise its cumulative expected reward over time. While UCB1 does not assume knowledge of the exact expected values of the arms, it makes some assumptions about the distribution from which these rewards are drawn. It is an algorithm to handle the exploration-exploitation trade-off when taking decisions under uncertainty. *Exploration* means that the learning agent should gather more information about the available arms (actions), whereas *exploitation* is the process of selecting the arm with the highest estimated return. The selection strategy for the best action is governed by: $a_t = \underset{a}{\operatorname{argmax}} \left[Q_t(a) + c\sqrt{\frac{2\log t}{k_t(a)}} \right]$, where $Q_t(a)$ is the estimated reward of action a at iteration t, c controls the exploration-exploitation balance, and $k_t(a)$ is the number of times action a has been selected. Using this equation, the UCB1 will typically begin by attempting each action at least once and then start concentrating on the actions with the best score.

Although UCB1 was originally studied without the exploration constant c (assuming $c = 1$) [3], it is commonly used with this hyperparameter in practice due to the need to control the exploration-exploitation trade-off in different environments and problem settings. Several studies have shown the effectiveness of applying the c parameter with the UCB1 [19, 34, 22]. Therefore, in this paper, we focus on dynamically learning it. Additionally, for non-stationary environments, we explore the adaptation of other hyperparameters, such as exploration constant α, discount factor γ, and the window size for Sliding-window UCB, Discounted UCB, and Mean f-dsw TS [11, 5].

2.1 Dynamic Bayesian Optimisation for Multi-Arm Bandits (DBO-MAB)

Before discussing the details of our proposed DBO-MAB approach, we introduce a simulation to serve as a consistent example throughout our explanation. In this simulation, we model a system where each arm corresponds to a specific server composition (configuration), resulting in different response times. These response times, with varying means and variances based on the server compositions, are then used to derive the rewards. The primary objective of this simulation is to identify the server composition (or arm) that yields the highest cumulative reward by dynamically optimizing the exploration constant. Essentially, we aim to pinpoint the best exploration constant that maximizes the probability of selecting this optimal server composition. Central to our problem setting is the cost function, denoted as $f(c)$. This function represents the response time or similar metric that we aim to minimize in the given context, such as a web server's performance optimisation. The exploration constant c within our algorithm directly influences this cost function, and our goal is to find the value of c that minimizes $f(c)$. In different contexts, the cost function could also take the form of a reward function, where the objective would be to maximize it.

DBO-MAB is a meta-algorithm that dynamically adapts the exploration constant hyperparameter of the UCB1 bandit algorithm in particular and other MAB algorithms in general. It consists of two key phases: initializing UCB1 with different exploration constant values and evaluating their performance, followed by a Bayesian optimisation process to refine the exploration constant based on observed rewards.

UCB1 with Dynamic Exploration: At the outset, the learning agent is introduced to a specific workload (environment). To initiate the exploration, define the search space boundaries by sampling from a uniform distribution. This sets the UCB1 algorithm's initial hyperparameters. As the agent starts its exploration, each exploration constant value is tested for a span of z time steps to estimate the expected rewards. In our web server example, the reward mechanism is designed such that actions leading to lower means offer higher rewards (*reward signal:* $-cost$).

Bayesian optimisation (BO): In this phase, we apply the BO technique to find an optimal hyperparameter value (or set of values, in the case of multiple hyperparameters) for the MAB algorithm. This process aims to improve the running system by fitting it with new, optimized hyperparameter(s). The MAB agent then runs with these new input(s) to update its process and improve its overall behavior. We denote the search space of possible hyperparameter values as S, and the history of sampled values and their corresponding performance as H. Figure 1 depicts this process, where BO analyzes the reward feedback (represented as 'Cost') to dynamically adjust the hyperparameter values. This section elaborates on the components of BO and their roles in refining the agent's decision-making process.

Gaussian Process (GP): The statistical model used in BO to approximate our unknown objective function is the Gaussian process.

The GP is defined by two components [32] (i) a mean function $m(s)$ that gives the expected value for $f(s)$. This represents our current best guess. (ii) A covariance function $k(s, s')$ that defines the covariance between points. This captures uncertainties. We denote Gaussian process as: $f(s) \sim \mathcal{GP}(m(s), k(s, s'))$. The Gaussian process predicts $f(s')$ at unseen points s'. Its posterior distribution captures what we estimate (mean) and our uncertainty (variance) at s'. Crucially, the GP adapts its surrogate model as we sample more points. By continuously updating its posterior, it refines its approximation of $f(s)$ based on new data.

Acquisition Functions: To propose the candidate points inside the search space we use metric functions (acquisition functions) which estimate which value of the parameter may return the function's best result [23].

Minimising the acquisition function to determine the next sample point is the objective. More formally, the objective function $f(s)$ will be sampled at $s_t = \text{argmin}_s acq(s|H_{t-1})$, where acq is the acquisition function and $H_{t-1} = \{(s_1, f_1), ..., (s_{t-1}, f_{t-1})\}$ refers to the $t-1$ samples drawn from the objective function $f(s)$ so far. Popular acquisition functions are the Probability of Improvement (PI), Expected Improvement (EI) and Lower Confidence Bound (LCB). For our analysis, we will use the Lower Confidence Bound (LCB) as it explores more diverse input space regions and tries to guide the search from an optimistic perspective [35]. The corresponding formulation is $LCB(s) = \mu(s) - \beta\sigma(s)$, where $\mu(s)$ and $\sigma(s)$ are the mean and the standard deviation, respectively, of the GP posterior predictive at s, and β is the coefficient factor.

As shown in Algorithm 1, we sample an initial sample size n of hyperparameter values s from a distribution (for instance, uniform), and each point leads to a certain cost. In each iteration of the process, the learning agent adds a new point, which is determined through the Bayesian optimisation (BO) process, and removes the oldest point. However, some M pillar points (chosen with uniform spacing) are kept fixed and revisited during the process, ensuring they are never removed. These pillar points serve as reference points to help main-

tain diversity and stability in the search space. As the agent continues learning, we re-evaluate those pillar points every x time steps to update their values, allowing the optimisation process to incorporate new information and adapt accordingly. Moreover, for every new point introduced from the GP, the oldest non-pillar point is discarded. Notably, the algorithm does not return a final best point but continually updates the model and hyperparameters dynamically.

Our optimisation task has two major elements: the predicted objective function $f(s)$ (for predicting the unknown function $f(.)$) and a set H, which represents the set of values that we have so far. This set H is updated during the algorithm's execution by adding new points and removing old ones. At each time step t, the algorithm chooses an exploration constant $s_t \in S$ as a new input $f_t(s_t)$, which is computed based on the predicted objective function defined over S. The goal of the learning algorithm is to choose a sequence of decisions $s_1, s_2, ...$ such that $f_t(s_t)$ can be minimised. In other words, the algorithm aims to achieve a low cost $f(s^*)$ after T rounds, where s^* is the optimal point that the system has experienced so far that minimizes the objective function $f_t(s)$ over S. At iteration t, given the previous observation that $H_{t-1} = \{(s_1, f(s_1)), \ldots, (s_{t-1}, f(s_{t-1}))\}$, function f is inferred at points $s \notin H_{t-1}$ (which have not yet been tried out) via Gaussian Processes. Then, decide on the next point s_t via BO and finally run the MAB again with the updated hyperparameters. Figure 2 shows how the model attempts to predict the function over certain candidates of inputs s.

Dynamic Range Adjustment Using Inter-Quartile Mean (IQM): To focus the search on promising regions of the input space, DBO-MAB dynamically adjusts the range for s, based on the distribution of observed rewards or costs. Let H denote the set of hyperparameter values that have been evaluated so far, and $R(s)$ denote the corresponding set of observed rewards/costs for these hyperparameter values. The process for dynamically adjusting the search range is as follows: (i) Identify the first quartile (Q_1) and third quartile (Q_3) of the rewards/costs $R(s)$, (ii) from $R(s)$, select the rewards/costs R_{IQR} that lie within the IQR, and correspondingly select the subset of hyperparameter values $H_{\text{IQR}} \subset H$ associated with R_{IQR}, (iii) and compute the Inter-Quartile Mean (IQM) as the mean of the hyperparameter values in H_{IQR} $\text{IQM} = \frac{1}{|H_{\text{IQR}}|} \sum_{s \in H_{\text{IQR}}} s$. We will refer to our approach as BO-MAB when this dynamic adjustment is not applied.

Multiple Hyper-parameter Optimisation: When dealing with our MAB algorithms that have multiple hyper-parameters, our DBO-MAB approach applies the optimisation process independently for each hyper-parameter.

3 Experimental Results

In this section we assess our agent's effectiveness using simulations of a webserver environment. Faster response times correspond to higher rewards for the agent, as they indicate better server performance. The agent's goal is to maximize cumulative reward by minimizing response times, which are treated as costs. This forms the core of our analysis where we investigate the agent's decision-making prowess and its adaptability across varying environments. For each action i, the associated cost is modeled by a normal distribution N$\sim(\mu_i, \sigma_i)$. Consequently, in our experimental framework, the reward is inversely related to the cost, defined as the negative of the cost ($-cost$), implying that a lower cost yields a higher reward for the agent. Hence, the agent's pursuit to minimize this cost can be equivalently seen as its endeavor to maximize the cumulative reward. The effectiveness of the proposed method, alongside other baseline approaches, is evaluated through two primary metrics: the cumulative average response time (CART) and the probability of selecting the best action (P_{best}). Specifically, we define CART for a given algorithm A as CART(A) = $\frac{1}{T} \sum_{t=1}^{T} r_t(A)$, where $r_t(A)$ represents the response time observed at time t and T is the total number of time steps. Additionally, the agent's efficiency in selecting the optimal action is quantified by the probability of selecting the best action (P_{best}), expressed as $P_{best}(A) = \frac{1}{T} \sum_{t=1}^{T} I(a_t^* = a_t(A))$, where a_t^* is the optimal action at time t, and $I(.)$ is the indicator function, which equals 1 if the action chosen by algorithm A at time t is the best possible action, and 0 if not.

We benchmark our agent against a gamut of baselines. Including UCB1 [3], parameter-free algorithms including Thompson sampling (TS) [6] and Bootstrapping Upper Confidence Bound (Boots-UCB), and scheduling algorithms PB2 [28] and PBT [16]. Our code is publicly available on GitHub[1].

For dynamic environments, where the reward distribution changes over time, we employ the following: (i) Sliding-Window UCB [11] with $windowsize = 100$ that captures the most recent actions and

Algorithm 1: DBO-MAB

Input: Number of iterations T, n, z, x

Initialize:

1. Sample points $s \sim \mathcal{U}$ n times, forming H
2. Run MAB with each point for z iterations
3. Select M uniform pillars points
4. Build initial GP model

for $t = 1$ **to** T:

1. Select next point to evaluate, s_t, using the GP model and the LCB acquisition function
2. Continue run MAB(s_t) for z iterations, compute $f(s_t)$
3. Update the GP model with $(s_t, f(s_t))$
4. Remove the oldest non-pillar point
5. Re-evaluate all selected M after x iterations
6. If Q1 and Q3 are defined:
   - Compute IQM
   - Dynamically adjust search range (s_t)

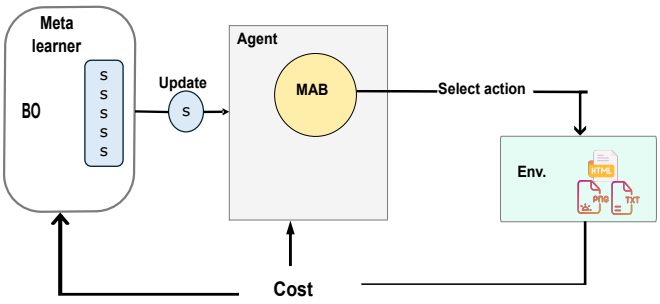


Figure 1: Architecture of the proposed DBO-MAB method. Meta Learner (Bayesian optimisation, BO) dynamically fine-tunes parameters for the MAB agent. The agent selects server configurations (actions). Env (Environment) refers to the workload pattern, which can be an image, text, or both, with varying sizes. The "Cost" represents the average response time for each point and the feedback that is used to enhance parameter selection iteratively.

[1] https://github.com/lsmcolab/DBO-MAB/tree/master

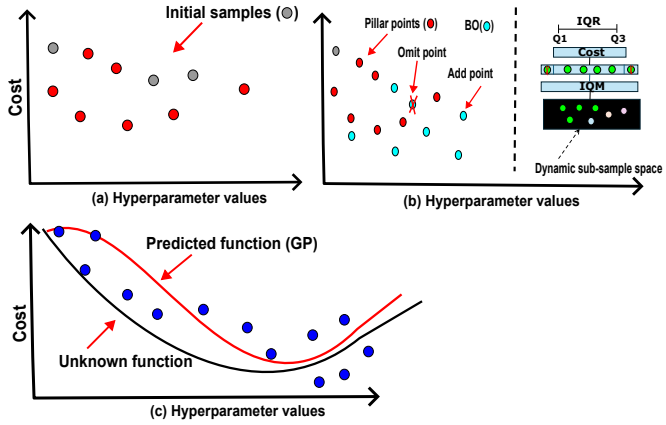
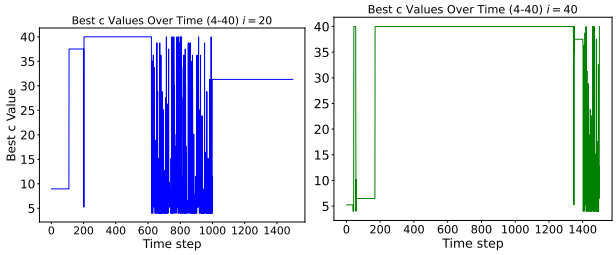


Figure 2: Evolution of the model with the incorporation of new samples of the hyper-parameter values s. (b) DBO-MAB initially uses the original range space and then adjusts the range based on IQM.



(a) Dynamic environment with i = 20. (b) Dynamic environment with i = 40.

Figure 3: Variation of optimal c values over time in a stochastic reward environment for a different number of actions i and c value range $4 - 40$.

rewards observed, and $\alpha = 1$ which represents the exploration constant, (ii) Discounted UCB [11] with $\gamma = 0.99$: discount factor with $0 < \gamma \leq 1$, $\alpha = 1$, (iii) Mean f-dsw TS (with mean as an aggregation function) [5] with $windowsize = 100$ and $\gamma = 0.9$, and (iv) Sliding-Window TS [29] with $windowsize = 100$.

We will start by studying how the optimal c value changes over time, before presenting the rest of our experimental analysis.

Analysis of Optimal c Values: Figure 3 illustrates the fluctuations in optimal c values over time, indicating that a static c parameter may not be universally applicable across different workloads. Instead, a dynamic approach to tuning c is necessary to optimize performance. The variability in the environment's reward distribution likely impacts the choice of the optimal c value.

Baselines: In the case of UCB1, the learning agent maintains a constant exploration factor (with a default value = 1) throughout its execution. Moreover, we applied the following baselines to adjust the UCB1 hyperparameters dynamically: PBT (Population Based Training), and PB2 (Population Based Bandit optimisation) is an extension of PBT that incorporates Bayesian optimisation into the process.

Environments: We tested our proposed online incremental learning algorithms in different static and dynamic random environments. In this simulation, these environments represent a variation of workload requests size using the Gaussian reward distributions with the parameters μ (mu) and σ (sigma). In each environment, the tasks take the form of MAP problems with multiple actions (arms) spaces $(20, 40, 60, 80, 100, 120, 140\ and\ 160)$ and the number of timesteps to 4000-10000, for every randomly generated environment. We will examine the following environments or workloads: (i) Diverse Mean Response Times with high Variance (random μ_s and σ_s scaled $0 - 1$); (ii) Small Mean Response Times with High Variance (random μ_s and σ_s scaled $0 - 0.1$). (iii) High Mean Response Times with High Variance.

Average response time workload between 0 and 1: Each action available to the learning agent is associated with a specific mean response time, denoted by the value of μ. The optimal action is the lowest μ, and the best method should assign a high probability of selecting actions with low μ. We define it as: μ_s range (0.2 - 0.7), σ_s range (0.1 - 0.25), optimal action: $\mu = 0.1$ with high $\sigma = 0.2$.

Average response time workload between 0 and 0.1: The characteristics of this environment are as follows: μ_s range (0.02 - 0.07), σ_s range (0.01 - 0.025), optimal action: $\mu = 0.01$ and ($\sigma = 0.02$).

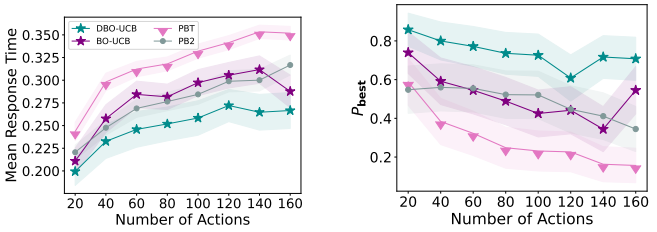
Average response time in a high workload environment: The characteristics of this environment are as follows: μ_s range $(60-98)$, σ_s range $(70 - 95)$. However, there is a dominant action with $\mu = 60.0$ and $\sigma = 95.0$, introducing a distinct reward distribution.

Baselines study: We will undertake a comprehensive analysis of the DBO-UCB (DBO-MAB applied to the UCB1 algorithm) strategy and its variant in contrast to several baseline methods. For workload between 0 and 1 and as depicted in Figure 4 and 5, there is a noticeable trend in the mean response time and the probability of selecting the best action, with DBO-UCB consistently outperforming other baselines. For scenarios involving 20 actions, DBO-UCB exhibits a $\approx$ 20% reduction in response time and around 95% for the P_{best} and for a higher number of actions ($i = 160$), it shows a constant performance by achieving $\approx$ 25% regarding to CART and $\approx$ 80% of P_{best}. For the same number of actions, BO-UCB (no dynamic range adjustment) demonstrates a similar behavior. In contrast, standard methodologies such as the UCB1 showed a significant drop in the P_{best}, from about 45% for $i = 20$ down to 10% for $i = 160$. Similarly, Thompson Sampling (TS) and Bootstrapping UCB (Boots-UCB), despite starting with high P_{best} values around 90%, experienced significant decreases as the number of actions increases, dropping to $\approx$ 20% and 15% respectively, for $i = 160$. Notably, Figure 5 reveals that within an environment with a reward range of $[0, 1]$, the UCB1 algorithm, with its default exploration parameter, fails to reach the level of efficiency demonstrated by the proposed method (DBO-UCB). This contrast demonstrates the value of dynamically learning the exploration parameter, hence improving decision-making.

For Figure 6 and 7 we show the results for the workload average response time $[0 - 0.1]$. Again, the DBO-UCB obtained the best-performing method which achieved the lowest average response time across all number of actions and $\approx$ 95% of the P_{best} and stays constantly above 80%. This approach maintains a response time that is nearly 22% lower than the PB2 method ($i = 20$) and around 18% lower for $i = 160$. In this environment, the BO-UCB shows good performance as the number of actions increases and scores a response time around 15% lower than the PBT for $i = 160$. Also, Thompson Sampling (TS), Bootstrapping UCB (Boots-UCB), and UCB1 show a decline in P_{best} when i is increasing, going from a $\approx$ 15%, at $i = 40$, to below 10%, at $i = 160$.

For a high workload environment with 160 actions (Figures 8a and 8b), UCB1 and Boots-UCB maintain a high response time with a low $P_{best} \approx 35\%$, whereas the DBO-UCB shows lower response time after an initial learning phase with higher P_{best} around 60%.

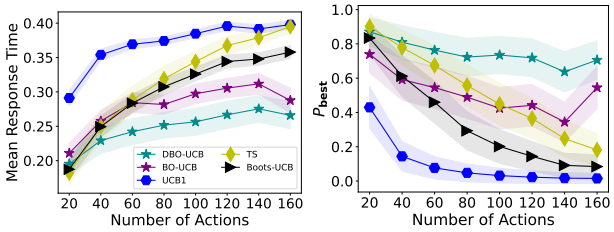
In addition, as we can see from Table 1, DBO-UCB showed a lower time budget, particularly when compared to Boots-UCB. DBO-UCB has a nearly 99% (Table 2) lower time budget than Boots-UCB, suggesting a reasonable computational cost for real-world problems.



(a) Average performance (lower is better) (baselines and proposed method)

(b) Probability of best action (higher is better)

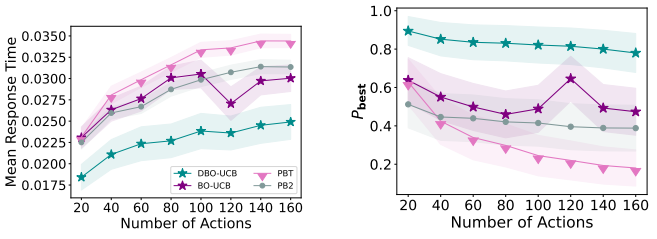
Figure 4: Comparing best-performing agent across multiple actions using different hyperparameter optimisation methods for average workload between 0-1.



(a) Average performance

(b) Probability of best action

Figure 5: Comparison of the proposed method against parameter-free strategies and UCB1 with c value of 1, across an average workload range of $0 - 1$.



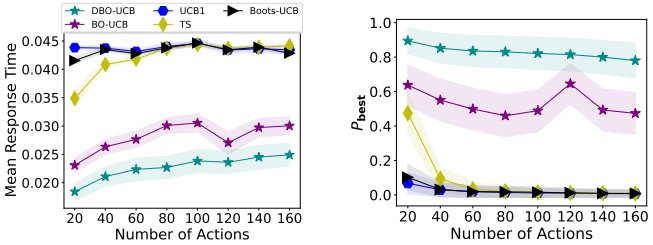
(a) Average performance

(b) Probability of best action

Figure 6: Comparing agents across multiple actions for average workload between $0 - 0.1$.

Average response time for dynamic environments: In this experiment, we evaluate the performance of our proposed method, DBO-UCB, and its variant in environments where the reward distributions are non-stationary (i.e., they change over time). The reward distributions remain constant over $\tau = 2000$ time steps before transitioning to a new distribution.

As shown in Figure 9a, initially, Mean d-sw TS and SW-TS are the best-performing algorithms, particularly for $i = 20$ and $i = 40$. However, as the number of actions increases, they experience a notable increase in their mean response time at around 0.35 for $i = 160$. In contrast, DBO-UCB demonstrates a more stable result as the number of actions increases and scored around 0.25 for $i = 160$.



(a) Average performance

(b) Probability of best action

Figure 7: Comparison between the proposed method, parameter-free methods, and the UCB1 across multiple actions for average workload between $0 - 0.1$.

Table 1: Computational Efficiency (s) for MAB

Method	Num. of Actions				
	80	100	120	140	160
DBO-UCB	439	470	441	466	490
UCB1	82	94	106	120	132
Boots-UCB	34253	36336	38455	41119	47840

For dynamic average workloads between 0 and 0.1 (Figure 11a), DBO-UCB's average response time is approximately 33% lower than Mean d-sw TS when the number of actions is $i = 20$, and this performance gap widens to around 45% lower than Mean d-sw TS for $i = 160$.

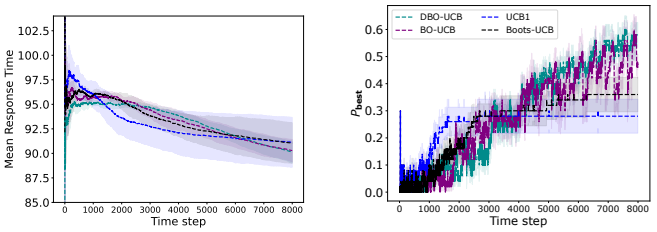
In this part, we extend the dynamic parameter optimization approach, foundational to the DBO-MAB method, to other bandit strategies. Specifically, we focus on learning the α parameter of both SW-UCB (DBO-SW-UCB) and Discounted UCB (DBO-Dis-UCB), and the discount factor γ and window size of the Mean d-sw TS (DBO-Mean d-sw TS) dynamically. As illustrated in Figure 10a, an improvement arises when we integrate the DBO-MAB approach into the SW-UCB. This dynamic optimization results in a noticeably lower response time around 14% lower than the standard SW-UCB for $i = 100$ and about 20% lower for $i = 160$. For an average workload [0-0.1] as shown in Figure 12a, achieved lower response time around 40% for $i = 20$ and 50% for $i = 160$.

DBO-Dis-UCB (Figure 12a) achieves a lower response time 28% for $i = 20$ and around 35% with $i = 160$ compared to standard Discounted UCB. In contrast, the DBO-Mean d-sw TS displays a moderate improvement in P_{best}.

Here, we extend the proposed method's ability to dynamically adjust various parameters of Multi-Armed Bandit (MAB) algorithms. For instance, as shown in Figure 13, DBO-SW-UCB dynamically adjusts the α parameter and $windowsize$. The optimized algorithm outperforms SW-UCB, demonstrating $\approx 30\%$ lower mean response time for $i = 160$ in sudden and incremental environmental changes. For further results and the specific parameters used, you may refer to the technical appendix (https://github.com/lsmcolab/DBO-MAB/blob/master/Technical_Appendix.pdf).

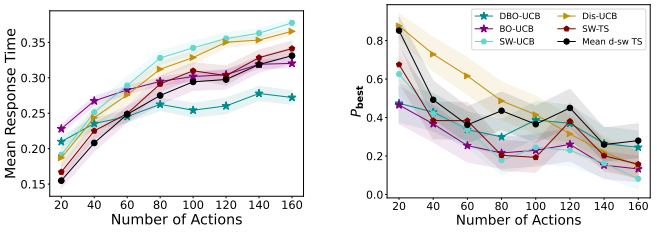
Table 2: Efficient Time Summary

Method	DBO-UCB %
BO-Update	**48.97%**
BO-Discard	-24.38%
BO-UCB	-4.11%
UCB1	-331.84%
Boots-UCB	**98.84%**



(a) Mean response time (b) Probability of best action

Figure 8: Comparison of the proposed method against parameter-free strategies and UCB1 with c default value, across an average workload range of $75 - 100$ (160 actions).



(a) Average performance (b) Probability of best action

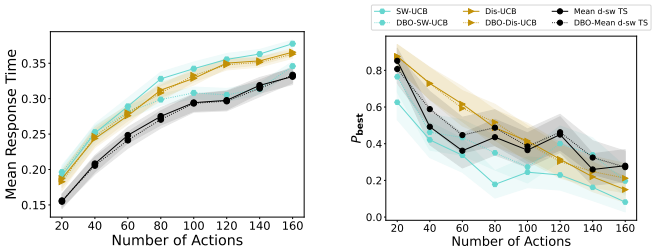
Figure 9: Comparing best performing agent across multiple actions for dynamic average workload between $0 - 1$.

Ablation study: To assess the efficacy of our proposed approach, we devised a series of experiments that included modifying aspects of the learning agent. The goal of these experiments was to determine how the changes to these aspects would affect the overall functionality of the system. There are two methods that have been studied:

(i) BO-Update method, which is an incremental approach that keeps every single point during the execution. It starts with some initial samples, then applies BO to select the new hyperparameter configurations for the models, and (ii) BO-Discard approach, distinct from other incremental learning methods, does not retain pillar points. Instead, it starts with initial samples and applies BO, with the model focusing on more current and potentially relevant data by discarding the oldest points during the optimisation process. In general, all methods perform well in most cases. However, as shown in Figure 14, DBO-UCB and its variant BO-UCB outperform the other methods, especially in more complex situations, such as workloads with small means (μ) and high variance (σ).

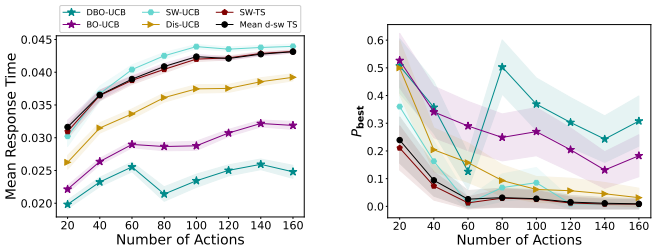
Though BO-Update and BO-Discard provide meaningful comparisons against the baselines, DBO-UCB and BO-UCB consistently outperform them by achieving a reduced response time and a higher likelihood of choosing the best action. For example, in the smaller workload environment with 120 actions, as illustrated in Figure 14b, BO-UCB is $\approx$ 40% more likely to select the optimal action than BO-Update and BO-Discard. For the same environment, DBO-UCB appears to be the best-performing method and achieves the maximum for lower actions ($i = 20$ with P_{best} of 90%), while it stays constantly above 80% for all the considered values of i.

Overall, it is evident that parameterization plays a significant role in the performance of the algorithms. Choosing the right parameterization method, like DBO-MAB in this case, can lead to better performance in terms of mean response time.



(a) Average performance (b) Probability of best action

Figure 10: Comparing best performing agent across multiple actions for dynamic average workload between $0 - 1$ applying DBO-UCB.



(a) Average performance (b) Probability of best action

Figure 11: Comparing best performing agent across multiple actions for dynamic average workload between $0 - 0.1$.

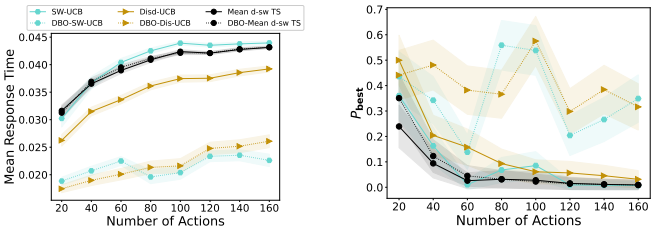
4 Related Works

Our approach intersects the concepts of continual learning and meta-learning. In this section, we discuss the prior research and current best practices for each of these areas.

Incremental/Continual learning: A significant amount of work on supervised continuous learning has been assessed, as a result of the growing interest in continuous learning research [27, 7, 13]. Most of the studies covered task-incremental learning scenarios, but the initial learning was followed by either restricted or complete lack of access to data from earlier tasks. In contrast, while our learning agent adapts sequentially in an online way, it is important for us to maintain the performance of the previous tasks, which may lead to fast convergence.

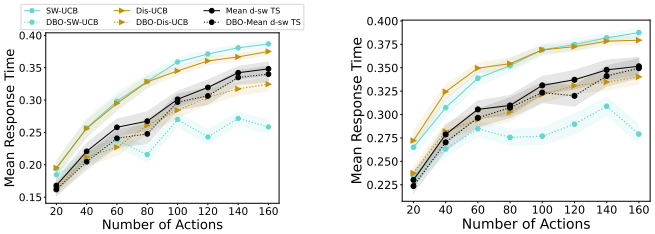
The study of [18] focused on continual reinforcement learning and its natural fit to the study of continuous learning and consequently, to lifelong improvement. Recent work by Kessler et al. [17] on continual learning introduced the OWL (*CO*ntinual *RL* *W*ithout *Conf*f*Lict*) method. In their approach, they framed the policy selection as a multi-armed bandit (MAB) problem during the test time to choose the optimal strategy for achieving the highest reward on a particular test task. Furthermore, the authors highlighted that sequential learning may be susceptible to catastrophic forgetting and interference. However, the DBO_MAB method is adept at retaining and building upon past experiences.

Bayesian optimisation: In the field of optimisation, Bayesian optimisation is a powerful technique for addressing problems with expensive and noisy evaluations. It is particularly useful in scenarios where the function to be optimized is expensive to evaluate, such as in reinforcement learning (RL) where evaluating a policy can require many episodes of interaction with the environment. Bayesian optimisation works by building a probabilistic model of the function to be optimized, and updating it as new data is acquired. The probabilis-



(a) Average performance (b) Probability of best action

Figure 12: Comparing best performing agent across multiple actions for dynamic average workload between $0 - 0.1$ by applying DBO-UCB method.



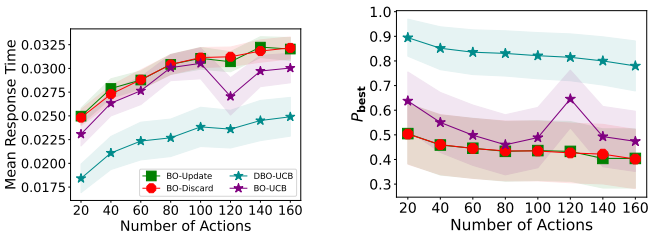
(a) Sudden changes (b) Incremental changes

Figure 13: Scalability of MAB algorithms under sudden and incremental environmental changes.

tic model, typically a Gaussian process, is used to make predictions about the function's behavior, and an acquisition function is used to determine the next point to evaluate. The acquisition function balances exploration, to improve the model's understanding of the function, and exploitation, to find the optimal point [4, 26].

Meta-learning: This is rapidly growing in importance as an area of machine learning research. Meta-learning, or the notion of learn-to-learn [21], is a branch of machine learning that autonomously changes an existing learning mechanism based on previous experiences and prior knowledge to swiftly learn new tasks [24]. Assuming that new tasks are connected to prior tasks, cumulative knowledge should be acquired in such a way that it retains the common structure of previously learned tasks [2]. As our approach is conducted in realtime we mainly focus on online meta-learning, in which tasks occur sequentially at the runtime, and on finding a way to learn from past experience and then apply that to the new task to avoid learning from scratch. Meta-learning is comparable to multi-task learning in that the learner has to perform a large number of tasks. Nonetheless, meta-learning requires the learning process to prepare for any new tasks while, as in multi-task learning, the learner is presented with a set number of tasks and is able to discover the relationships between them [24].

A recent research [12] showed that meta-RL algorithms benefit from prior learning to gain proficiency in learning new tasks rapidly. However, to perform this approach, a huge amount of meta-training tasks must be offered to the meta-learning. Meta-learning tries to discover which learning approaches perform best on various types of data [30]. In sum, we acquire the ability to learn throughout tasks. However, the difficulty with meta-learning is that it must be done in a methodical, data-driven manner. To begin, we must collect meta-data describing previously learned tasks and models. They include the precise configurations of the algorithms that are used for model training, particularly hyper-parameter values. Second, we must ex-



(a) Average performance (b) Probability of best action

Figure 14: Comparing best performing agent across multiple actions for static average workload between $0 - 0.1$.

tract and transmit information from these past meta-data to drive the search for optimum models for the new tasks [31].

Meta-learning can be approached in a variety of ways, resulting in a variety of possible applications, including Reinforcement Learning [9, 25, 10], acquiring knowledge from a related task, transfer of meta-knowledge between tasks and recommendation of items [31]. Azar et al. [20] investigated sequential multitask learning in MABs.

Moreover, Duan et al. (2016) [8] proposed a comprehensive strategy for strengthening RL that involves the task-specific rapid Reinforcement Learning algorithms led by the slow meta-RL approach. Wang et al. (2016) [33] suggested using a deep RL algorithm to train an RNN on the previous interval's actions and rewards for it to learn a base-level reinforcement learning algorithm for specific tasks. Rather than employing tasks that are relatively unstructured, such as random MDPs, they concentrate on task distributions that are structured (e.g., dependent bandits), allowing the meta-RL method to leverage the underlying task structure.

The recent method PB2 (Population-Based Bandit) [28] for online hyperparameter optimisation focuses on scheduler hyper-parameter optimisation and parallel execution in RL setting, and it uses the Bayesian optimisation to select the hyperparameters for the next generation of models in the population. This leads to a more efficient search of the hyperparameter space, which in turn results in better performance. Another notable work in the context of UCB algorithms is the Bootstrapped UCB (Bootstrapping Upper Confidence Bound) [14]. That paper proposes a non-parametric, data-dependent UCB algorithm based on multiplier bootstrap. The Bootstrapped UCB method focuses on constructing tighter confidence bounds using multiplier bootstrap to improve exploration and exploitation in the context of bandit problems. While our DBO-MAB approach combines Bayesian optimisation with MAB algorithms.

5 Conclusion

In this work, we considered the challenge of an unknown reward range in static and dynamic environments. The DBO-MAB algorithm continuously learns effectively and maintains its performance on previous tasks. This approach utilizes the MAB algorithms and Bayesian optimisation (BO) to select new hyperparameter configurations for the models, allowing for more efficient exploration of the hyperparameter space dynamically. We evaluated our method on various workloads, and the results demonstrated significant performance improvements compared to existing approaches. Through optimizing the hyperparameter values dynamically, the algorithm was able to achieve better results more quickly, enhancing its overall efficiency.

Acknowledgments

This work was supported by the Saudi Arabian Ministry of Education, the Saudi Arabian Cultural Bureau in London, and Qassim University. We also wish to thank the High-End Computing cluster support team at Lancaster University. We would like to thank Dr. Matheus Alves and Gao Peng for their valuable feedback.

References

[1] R. Agarwal, M. Schwarzer, P. S. Castro, A. C. Courville, and M. Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *Advances in neural information processing systems*, 34: 29304–29320, 2021.

[2] R. Amit and R. Meir. Meta-learning by adjusting priors based on extended pac-bayes theory. In *International Conference on Machine Learning*, pages 205–214. PMLR, 2018.

[3] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2):235–256, 2002.

[4] J. C. Barsce, J. A. Palombarini, and E. C. Martínez. Towards autonomous reinforcement learning: Automatic setting of hyperparameters using bayesian optimization. In *2017 XLIII Latin American Computer Conference (CLEI)*, pages 1–9. IEEE, 2017.

[5] E. Cavenaghi, G. Sottocornola, F. Stella, and M. Zanker. Non stationary multi-armed bandit: Empirical evaluation of a new concept drift-aware algorithm. *Entropy*, 23(3):380, 2021.

[6] O. Chapelle and L. Li. An empirical evaluation of thompson sampling. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, NIPS'11, page 2249–2257, Red Hook, NY, USA, 2011. Curran Associates Inc. ISBN 9781618395993.

[7] M. De Lange, R. Aljundi, M. Masana, S. Parisot, X. Jia, A. Leonardis, G. Slabaugh, and T. Tuytelaars. Continual learning: A comparative study on how to defy forgetting in classification tasks. *arXiv preprint arXiv:1909.08383*, 2(6):2, 2019.

[8] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel. Rl^2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

[9] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pages 1126–1135. PMLR, 2017.

[10] C. Finn, K. Xu, and S. Levine. Probabilistic model-agnostic meta-learning. *arXiv preprint arXiv:1806.02817*, 2018.

[11] A. Garivier and E. Moulines. On upper-confidence bound policies for non-stationary bandit problems. *arXiv preprint arXiv:0805.3415*, 2008.

[12] A. Gupta, B. Eysenbach, C. Finn, and S. Levine. Unsupervised meta-learning for reinforcement learning. *arXiv preprint arXiv:1806.04640*, 2018.

[13] R. Hadsell, D. Rao, A. A. Rusu, and R. Pascanu. Embracing change: Continual learning in deep neural networks. *Trends in cognitive sciences*, 24(12):1028–1040, 2020.

[14] B. Hao, Y. Abbasi Yadkori, Z. Wen, and G. Cheng. Bootstrapping upper confidence bound. *Advances in neural information processing systems*, 32, 2019.

[15] E. Hazan. Introduction to online convex optimization. *arXiv preprint arXiv:1909.05207*, 2019.

[16] M. Jaderberg, V. Dalibard, S. Osindero, W. M. Czarnecki, J. Donahue, A. Razavi, O. Vinyals, T. Green, I. Dunning, K. Simonyan, et al. Population based training of neural networks. *arXiv preprint arXiv:1711.09846*, 2017.

[17] S. Kessler, J. Parker-Holder, P. Ball, S. Zohren, and S. J. Roberts. Same state, different task: Continual reinforcement learning without interference. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 7143–7151, 2022.

[18] K. Khetarpal, M. Riemer, I. Rish, and D. Precup. Towards continual reinforcement learning: A review and perspectives. *arXiv preprint arXiv:2012.13490*, 2020.

[19] L. Kocsis and C. Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.

[20] A. Lazaric, E. Brunskill, et al. Sequential transfer in multi-armed bandit with finite set of models. *Advances in Neural Information Processing Systems*, 26, 2013.

[21] J. Li and M. Hu. Continuous model adaptation using online meta-learning for smart grid application. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[22] N. Manome, S. Shinohara, and U.-i. Chung. Simple modification of the upper confidence bound algorithm by generalized weighted averages. *arXiv preprint arXiv:2308.14350*, 2023.

[23] O. Martin. *Bayesian Analysis with Python: Introduction to statistical modeling and probabilistic programming using PyMC3 and ArviZ*. Packt Publishing Ltd, 2018.

[24] D. Meunier and P. Alquier. Meta-strategy for learning tuning parameters with guarantees. *arXiv preprint arXiv:2102.02504*, 2021.

[25] N. Mishra, M. Rohaninejad, X. Chen, and P. Abbeel. A simple neural attentive meta-learner. *arXiv preprint arXiv:1707.03141*, 2017.

[26] S. Müller, A. von Rohr, and S. Trimpe. Local policy search with bayesian optimization. *Advances in Neural Information Processing Systems*, 34:20708–20720, 2021.

[27] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71, 2019.

[28] J. Parker-Holder, V. Nguyen, and S. J. Roberts. Provably efficient online hyperparameter optimization with population-based bandits. *Advances in Neural Information Processing Systems*, 33:17200–17211, 2020.

[29] F. Trovo, S. Paladino, M. Restelli, and N. Gatti. Sliding-window thompson sampling for non-stationary settings. *Journal of Artificial Intelligence Research*, 68:311–364, 2020.

[30] J. N. van Rijn, G. Holmes, B. Pfahringer, and J. Vanschoren. The online performance estimation framework: heterogeneous ensemble learning for data streams. *Machine Learning*, 107(1):149–176, 2018.

[31] J. Vanschoren. Meta-learning: A survey. *arXiv preprint arXiv:1810.03548*, 2018.

[32] N. A. Vien, H. Zimmermann, and M. Toussaint. Bayesian functional optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[33] J. X. Wang, Z. Kurth-Nelson, D. Tirumala, H. Soyer, J. Z. Leibo, R. Munos, C. Blundell, D. Kumaran, and M. Botvinick. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016.

[34] M. Xie, W. Yin, and H. Xu. Autobandit: A meta bandit online learning system. In *IJCAI*, pages 5028–5031, 2021.

[35] S. Zhang, F. Yang, C. Yan, D. Zhou, and X. Zeng. An efficient batch-constrained bayesian optimization approach for analog circuit synthesis via multiobjective acquisition ensemble. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(1):1–14, 2021.