Tuning Mixed Input Hyperparameters on the Fly for Efficient Population Based AutoRL

Anonymous Author(s)
Affiliation
Address
email

Abstract

Despite a series of recent successes in reinforcement learning (RL), many RL algorithms remain sensitive to hyperparameters. As such, there has recently been interest in the field of AutoRL, which seeks to automate design decisions to create more general algorithms. Recent work suggests that population based approaches may be effective AutoRL algorithms, by learning hyperparameter schedules on the fly. In particular, the PB2 algorithm is able to achieve strong performance in RL tasks by formulating online hyperparameter optimization as time varying GP-bandit problem, while also providing theoretical guarantees. However, PB2 is only designed to work for *continuous* hyperparameters, which severely limits its utility in practice. In this paper we introduce a new (provably) efficient hierarchical approach for optimizing *both continuous and categorical* variables, using a new time-varying bandit algorithm specifically designed for the population based training regime. We evaluate our approach on the challenging Procgen benchmark, where we show that explicitly modelling dependence between data augmentation and other hyperparameters improves generalization.

1 Introduction

Reinforcement Learning (RL [64]) is a paradigm whereby agents learn to make sequential decisions through trial and error. In the past few years there have been a series of breakthroughs using RL in games [59, 38, 7] and robotics [42, 30], which have led to a surge of interest in the machine learning community. Beyond these examples, RL offers the potential to impact vast swathes of society, from autonomous vehicles to robotic applications in healthcare and industry.

Despite the promising results, RL is notoriously difficult to use in practice. In particular, RL algorithms are incredibly sensitive to hyperparameters [21, 2, 17, 25, 10], with careful tuning often the difference between success and failure. Furthermore, as new and more complex algorithms are introduced, the search space continues to grow [2]. As a result, many lauded approaches are impossible to reproduce without the vast resources required to sweep through hundreds or thousands of possible configurations [6]. This makes it almost impossible to apply these methods in novel settings, where optimal hyperparameters are unknown. On the other hand, the capability of current methods may be understated, as better configurations could boost performance [11, 73].

In this work we focus on the recent impressive results for *Population Based Training* (PBT, [27, 36]), which has demonstrated strong performance in a variety of prominent RL settings [55, 73, 37, 18, 26]. PBT works by training agents in parallel, with an evolutionary outer loop optimizing hyperparameters, periodically replacing weaker agents with perturbations of stronger ones. However, since PBT relies on random search to explore the hyperparameter space, it requires a large population size which can be prohibitively expensive for small and even medium sized labs.

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

Table 1: The components of related approaches, and their relative trade-offs.

Algorithm	Population Based?	Model Based?	Categorical Variables?
PBT/PBA [27, 23]	✓	✗	●
PB2 [44]	✓	✓	✗
CoCaBO [53]	✗	✓	✓
This work	✓	✓	✓

In order to achieve similar success with a smaller computational budget, the recent *Population Based Bandits* (PB2, [44]) algorithm improved sample efficiency by introducing a probabilistic exploration step, backed by theoretical guarantees. Unfortunately, a key limitation of PB2 is that it only addresses the problem of efficiently selecting *continuous* hyperparameters, inheriting the random search method for *categorical* variables from the original PBT. This is not only inefficient but also crucially ignores potential dependence between continuous and categorical variables. Since the GP model is completely unaware of the changing categories, it is unable to differentiate between trials that may have completely different categorical hyperparameters. In this paper we introducing a new hierarchical approach to PB2, which can efficiently model *both* continuous and categorical variables, with theoretical guarantees. Our main contributions are the following:

Technical: We introduce a new PB2 explore step which can *efficiently* choose between *both* categorical and continuous hyperparameters in a population based training setup. In particular, we propose a new time varying batch multi-armed bandit algorithm, and introduce two hierarchical algorithms which condition on the selected categorical variables. We show our new approach achieves sublinear regret, extending the results for the continuous case with PB2.

Practical: We scale our approach to test generalization on the Procgen benchmark [14], an active area of research. We demonstrate improved performance when explicitly modelling dependence between data augmentation type and continuous hyperparameters (such as learning rate) vs. baselines using random search to select the data augmentation.

2 Related Work

This work contributes to the emerging field of AutoRL [12], which seeks to automate elements of the reinforcement learning (RL) training procedure. Automating RL hyperparameter tuning has been studied since the 1980s [4], with a surge in recent interest due to the increasing complexity of modern algorithms [73, 46]. The scope for AutoRL is broad, from using RL in novel real-world problems where hyperparameters are unknown [54], to improving the performance of existing methods [11]. Recent successes in AutoRL include learning differentiable hyperparameters with meta-gradients [70, 72], while it has recently been shown it is even possible to learn algorithms [41, 13].

In this paper we focus on the class of *population based training* (PBT [27]) methods. Inspired by how humans run experiments, PBT trains agents in parallel and periodically replaces the weakest agents with variations of the stronger ones, learning a hyperparameter *schedule* on the fly, in a single training run. While PBT is applicable in any AutoML [24] setting, it has been particularly impactful in deep RL [18, 73, 55]. PBT was recently shown to work well with a shared replay buffer [19], but this only applies for off policy methods. This paper builds on the recently introduced *Population Based Bandits* (PB2, [44]) algorithm, which employs a Bayesian Optimization (BO, [9, 22, 60, 16, 1, 53]) approach to select hyperparameters during the course of training. Concretely PB2 casts the hyperparameter selection step of PBT as a batch time-varying GP bandit optimization problem [8]. The strength of PB2 lies in finding optimal configurations with a small population size, yet at present it has only been developed for continuous hyperparameters, relying on random search for the categorical variables.

Categorical variables are prominent in RL, for example the choice of exploration strategy [66], or even algorithm class. A prominent recent categorical variable is the choice of data augmentation, which has been shown to significantly improve efficiency and generalization in RL [34, 32, 35, 69], reducing observational overfitting [62]. Recent work introduced *automatic data augmentation* [48], which shows significant improvement over a static baseline by learning the data augmentation on the fly (with a single agent). However, this approach relies on a grid search over new hyperparameters, keeping all others fixed. We take inspiration from this result and learn *both* data augmentation and baseline RL algorithm hyperparameters jointly on the fly, with a population of agents. PBT has also been extended to data augmentation (PBA, [23]) with strong results in supervised learning tasks. However, in this setting the data augmentation was setup as a continuous variable. A high level summary of differences vs. prior work is shown in Table 1.

3 The Case for AutoRL

In this section we introduce the reinforcement learning (RL, [64]) paradigm, before making the case for automating hyperparameter selection for policy gradient algorithms.

3.1 Reinforcement Learning Background

A Markov Decision Process (MDP) is a tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, where for each time step $t = 0, 1, 2 \dots$ the environment provides the agent with an observation $s_t \in \mathcal{S}$, the agent responds by selecting an action $a_t \in \mathcal{S}$, and then the environment provides the next reward r_t, discount γ_{t+1}, and state s_{t+1}. Reinforcement learning (RL) considers the problem of learning a policy π which selects actions that maximize the expected total discounted reward [5, 65, 64]. While our method can be applied to **any** RL algorithm, for simplicity we focus our discussion on *policy gradient* algorithms, which directly seek to maximize the policy with respect to expected reward. For a policy $\pi : \mathcal{S} \to \mathcal{A}$ parameterized by θ the objective is as follows:

$$J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta} R(\tau)$$

where $\tau = \{s_1, a_1, r_1, \dots, s_H, a_H, r_H\}$ for some horizon H. Of course, we wish to maximize this, so optimize the policy by taking the following gradient steps:

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta J(\pi_{\theta_t})$$

3.2 Case study: Proximal Policy Optimization

Proximal Policy Optimization (PPO [58]) is one of the most widely used RL algorithms, achieving strong performance in continuous control problems, and even scaling to large scale games [7]. Building on Trust Region Policy Optimization (TRPO [56]), the success of PPO comes from using a clipped loss function as follows:

$$\mathcal{L}_{\text{PPO}}(\theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_\mu(a|s)} A^{\pi_\mu}, g(\theta, \mu) A^{\pi_\mu}\right), \text{ where } g(\theta, \mu) = \text{clip}\left(\frac{\pi_\theta(a|s)}{\pi_\mu(a|s)}, 1 - \epsilon, 1 + \epsilon\right)$$

for a previous policy π_μ, an advantage function A and a clipping hyperparameter ϵ. As can be seen, PPO's success hinges on several new hyperparameters. As well as the learning rate α, clip parameter ϵ, decay γ, there is also often another hyperparameter in the advantage function λ [57]. As if this wasn't enough, a recent study showed we also need to consider *code level* hyperparameters [17] after recent work found nine implementation details (such as reward scaling) had a significant impact on performance. Thus, it is clear that while PPO can be effective, it requires careful tuning if it is to be used effectively for novel problems.

In addition, a recent study investigating actor-critic algorithms [2] found that not only are there many more hyperparameters than previously thought (for example weight initialization strategy), but their optimal values are dependent on one another, producing a combinatorially large search space. This problem is only increasing as novel methods extending upon PPO they almost always add *additional* hyperparameters [48]. Thus, if it wasn't enough that PPO itself has tens of hyperparameters, new additions may render all previous values sub-optimal since they may be dependent on the new variables. The main hypothesis in this paper is that we can improve performance of baseline algorithms by *jointly* learning several hyperparameters *on the fly*.

4 Population Based Bandits

We consider the problem of optimizing a population of agents in parallel, dynamically adapting their weights and hyperparameters such that strong performance can be achieved in a single training run. Following existing work [27, 44], we consider two sub-routines, explore and exploit. We train for a total of T steps, evaluating performance every $t_{\text{ready}} < T$ steps. For the exploit step, the weights of the bottom agents are replaced by those from a randomly sampled agent from the set of best performing agents, in what is called *truncation selection*. The next step is to select new hyperparameters, with the explore step. The full procedure is shown in Algorithm 1 and Fig. 1.

Algorithm 1 Population Based Training (Bandits)

Initialize: Population network weights, hyperparameters, empty dataset.
for (in parallel) $t = 1, \ldots, T-1$ **do**
1. **Train Models**
2. **Evaluate Models and Record Data**
3. If $t \mod t_{\text{ready}} = 0$:
   - **Exploit:** Replace the weights of weaker agents with those from stronger agents.
   - **Explore:** If weights were replaced, select new hyperparameters.

Return the best trained model θ

The key difference between Population Based Training (PBT, [27]) and Population Based Bandits (PB2, [44]) is the explore step. PBT selects new hyperameters using random perturbations, which is effective with a large population size but performs poorly in a resource constrained setting. By contrast PB2 (blue in Alg. 1) stores data from all previous trials and formulates the online hyperparameter selection as a time-varying bandit optimization problem. In this paper we follow the approach from PB2, and extend it to consider efficient mechanisms for selecting categorical variables.

4.1 Online Hyperparameter Selection as GP-Bandit Optimization

We consider the problem of selecting optimal hyperparameters, x_t^b, from a compact, convex subset $\mathcal{D} \in \mathbb{R}^d$ where d is the number of hyperparameters. Here the index b refers to the bth agent in a population, and the subscript t represents the number of timesteps elapsed during the training of a neural network. In particular, we consider the *schedule* of optimal hyperparameters over time $\left(x_t^b\right)_{t=1,\ldots T}$. Let $F_t(x_t)$ be an objective function under a given set of hyperparameters at timestep t. Here we take $F_t(x_t)$ to be the reward for a deep RL agent. When training for a total of T steps, our goal is to maximize the final performance $F_T(x_T)$.

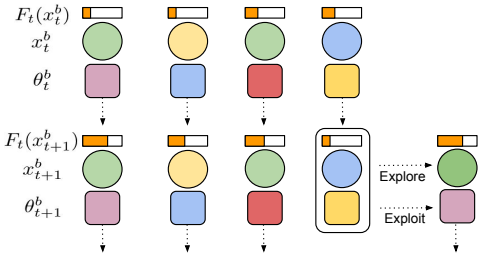


Figure 1: An overview of the PBT/PB2 framework. We have a population of B agents, each with weights θ_t^b and hyperparameters x_t^b. t refers to the stage of training, with each increment corresponding to training for t_{ready} steps. The performance of the agent is represented by F_t. At each t, the weights of the worst performing agents are replaced (exploit), where weights are copied from better agents. The new hyperparameters are then selected using an explore procedure, the focus of this work.

Following [44] we formulate this problem as optimizing the time-varying black-box reward function f_t, over $\mathcal{D}$. Every t_{ready} steps, we observe and record noisy observations, $y_t = f_t(x_t) + \epsilon_t$, where $\epsilon_t \sim \mathcal{N}(0, \sigma^2\mathbf{I})$ for some fixed σ^2. Note the function f_t represents the *change* in F_t after training for t_{ready} steps, i.e. $F_t - F_{t-t_{\text{ready}}}$. We define the best choice at each timestep as $x_t^* = \arg\max_{x_t \in \mathcal{D}} f_t(x_t)$, and so the **regret** of each decision as $r_t = f_t(x_t^*) - f_t(x_t)$. Minimizing the regret of each decision is equivalent to maximizing the final reward [44], i.e. $\max F_T(x_T) = \min \sum_{t=1}^{T} r_t(x_t)$. We now discuss how PB2 seeks to minimize regret for *continuous* hyperparameters.

4.2 Parallel Gaussian Process Bandits for a Time-Varying Function

PB2 models the data using a Gaussian Process (GP, [49]), thus assumes the data follows a Gaussian distribution with mean $\mu_t(x')$ and variance $\sigma_t^2(x')$ as:

$$\mu_t(x') \coloneqq \mathbf{k}_t(x')^T(\mathbf{K}_t + \sigma^2\mathbf{I})^{-1}\mathbf{y}_t \tag{1}$$

$$\sigma_t^2(x') \coloneqq k(x', x') - \mathbf{k}_t(x')^T(\mathbf{K}_t + \sigma^2\mathbf{I})^{-1}\mathbf{k}_t(x'), \tag{2}$$

where $\mathbf{K}_t \coloneqq \{k(x_i, x_j)\}_{i,j=1}^t$ and $\mathbf{k}_t \coloneqq \{k(x_i, x_t')\}_{i=1}^t$. The GP predictive mean and variance above will later be used to represent the exploration-exploitation trade-off in making decision under the presence of uncertainty. The key insight in PB2 is that it is possible to consider the time-varying nature of neural network hyperparameters. This is consistent with empirical results, for example, the use of learning rate schedules in almost all large scale neural network models since they have become

popular [33]. Concretely, they follow [8] by modeling the reward function under the time-varying setting as follows:

$$f_1(x) = g_1(x),$$
$$f_{t+1}(x) = \sqrt{1-\omega} f_t(x) + \sqrt{\omega} g_{t+1}(x) \quad \forall t \geq 2,$$

where $g_1, g_2, ...$ are independent random functions with $g \sim GP(0, k)$ and $\omega \in [0, 1]$ models how the function varies with time, such that if $\omega = 0$ we return to GP-UCB and if $\omega = 1$ then each evaluation is completely independent. This leads to the extensions of Eqs. (1) and (2) using the new covariance matrix $\tilde{\mathbf{K}}_t = \mathbf{K}_t \circ \mathbf{K}_t^{\text{time}}$ where $\mathbf{K}_t^{\text{time}} = [(1-\omega)^{|i-j|/2}]_{i,j=1}^T$ and $\tilde{\mathbf{k}}_t(x) = \mathbf{k}_t \circ \mathbf{k}_t^{time}$ with $\mathbf{k}_t^{time} = [(1-\omega)^{(T+1-i)/2}]_{i=1}^T$. Here $\circ$ refers to the Hadamard product.

Selecting hyperparameters for parallel agents. A key observation in [16] is that since a GP's variance (Eqn. 2) does not depend on y_t, the acquisition function can account for incomplete trials by updating the uncertainty at the pending evaluation points. Concretely, we define x_t^b to be the b-th point selected in a batch, after t timesteps. This point may draw on information from $t + (b-1)$ previously selected points. In the single agent, sequential case, we set $B = 1$ and recover $t, b = t - 1$. Thus, at the iteration t, we find a next batch of B samples $\left[x_t^1, x_t^2, ...x_t^B\right]$ by sequentially maximizing the following acquisition function:

$$x_t^b = \arg\max_{x \in \mathcal{D}} \mu_{t,1}(x) + \sqrt{\beta_t}\sigma_{t,b}(x), \forall b = 1, ...B \tag{3}$$

for $\beta_t > 0$. In Eqn. (3) we have the mean from the previous batch ($\mu_{t,1}(x)$) which is fixed, but can update the uncertainty using our knowledge of the agents currently training ($\sigma_{t,b}(x)$). This significantly reduces redundancy, as the model is able to explore distinct regions of the space.

To summarize, the PB2 explore step works by optimizing Eqn. (3) to select a batch of new parameters. However, it is clear to see that this approach is not equipped with a means to select *categorical* variables. In fact, the existing PB2 algorithm inherits the random selection of categories from PBT, and ignores this choice when selecting the continuous variables. To reflect this, from this point onwards we refer to PB2 from [44] as **PB2-Rand**.

5 Efficient Selection of Continuous and Categorical Variables

We propose a new class of hierarchical population based approaches for handling continuous and categorical variables. To the best of our knowledge, this is the first provably efficient approach for optimizing the categorical variables in the PBT family [27, 44]. First we introduce a new time-varying version of the parallel EXP3 algorithm, which we call TV.EXP3.M, used in all of our methods to select categories. We then present three alternative approaches to subsequently select the continuous variables in a hierarchical fashion, varying in their degree of dependence on the categorical choices.

5.1 Time-varying Parallel EXP3

We begin by presenting a new algorithm for parallel multi-armed bandits (MAB) under time-varying settings and adversarial feedback. At each round, multiple actions are taken among the set of all possible choices [67], making it possible to use this approach in a population of agents. In particular, we extend the multiple play EXP3 algorithm (or EXP3.M) [67] for modeling time-varying rewards. We summarize the high-level idea of this approach in Algorithm 2 and refer to Algorithm 5 and Sec. D.1 in the appendix for further details.

We treat each categorical choice as an arm in the bandit setting. Given a collection of C categories, each includes a reward probability, we select a batch of B points

$$A_t = [A_t^1, A_t^2, ...A_t^B] = \text{TV.EXP3.M}(D_{t-1}). \tag{4}$$

To efficiently select a set of B distinct arms from $[C]$, we use the technique of dependent rounding (DepRound) while satisfying the condition that each arm c is selected with probability p_c exactly [20]. TV.EXP3.M is provably efficient, with a sublinear regret bound.

Theorem 1. *Set $\alpha = \frac{1}{T}$ and $\gamma = \min\left\{1, \sqrt{\frac{C\ln(C/B)}{(e-1)BT}}\right\}$, we assume the reward distributions changes at arbitrary instances, but the total number of change points is no more than $V \ll \sqrt{T}$ times. The expected regret of TV.EXP3.M satisfies the following sublinear bound*

Algorithm 2 TV.EXP3.M brief

Input: $\gamma = \sqrt{\frac{C\ln(C/B)}{(e-1)BT}}$, $\alpha = \frac{1}{T}$, C #categorical choice, T #max iteration, B #multiple play

Init $w_c = 1, \forall c = 1...C$ and denote $\eta = (\frac{1}{B} - \frac{\gamma}{C})\frac{1}{1-\gamma}$
for $t = 1$ to T **do**
  Normalize w to prevent from over exploitation
  Compute the prob for each arm $p_t^c, \forall c$
  Select a batch $S_t = \text{DepRound}\left(B, \left[p_t^1 p_t^2 .... p_t^C\right]\right)$
  Observe the reward after evaluating S_t
  Update the weights using η for each arm $w_t^c, \forall c$
end for

Output: $\mathcal{D}_T$

$$\mathbb{E}\left[R_{TB}\right] \leq [1 + e + V]\sqrt{(e-1)\frac{CT}{B}\ln\frac{CT}{B}}.$$

5.2 New Exploration Strategies

Now we are ready to present two novel hierarchical exploration strategies. Both methods use Algorithm 2 to select categories before subsequently choosing continuous variables.

Algorithm 3 PB2-Mult: Explore

1. Select $\{h_t^b\}_{b=1}^B$ using TV.EXP3.M
2. Filter dataset.
3. Select x_t^b by optimizing Eqn.

Output: $\{h_t^b\}_{b=1}^B$, $\{x_t^b\}_{b=1}^B$

Algorithm 4 PB2-Mix: Explore

1. Select $\{h_t^b\}_{b=1}^B$ using TV.EXP3.M
2. Select x_t^b by optimizing Eqn. 5

Output: $\{h_t^b\}_{b=1}^B$, $\{x_t^b\}_{b=1}^B$

PB2-Mult: We consider a special version of dependency in which the presence of the continuous variables entirely depends on if the corresponding category is switched on. This not only allows us to fully incorporate dependence between hyperparameters, but also allows us to have different hyperparameters depending on the category. For example, if our categorical variable is the choice of neural network optimizer, then the current optimal learning rate may vary significantly if using Adam [31] or SGD. Furthermore, Adam requires additional hyperparameters, β_1 and β_2.

We handle this dependency by constructing multiple time-varying GP surrogate models. Each surrogate corresponds to a choice of category. We call this method **PB2-Mult**. PB2-Mult is the best choice if the continuous variables are highly dependent on the category, or if there are differing numbers of continuous variables for each category.

PB2-Mix: We also propose a mechanism to incorporate dependence between continuous and categorical variables *directly into the GP kernel*, in what we call **PB2-Mix**. To do this, we extend the joint continuous-categorical kernel in CoCaBO [53] to the time-varying setting. Denote $z = [x, h]$, we have:

$$k_z(z, z') = (1 - \lambda)\left(\tilde{\mathbf{k}}_t(x) + k_{ht}\right) + \lambda\tilde{\mathbf{k}}_t(x)k_{ht} \tag{5}$$

where $k_{ht} = \frac{\sigma_2}{C}\sum \mathbb{I}(h, h') \times \mathbf{k}_t^{time}$ and $\tilde{\mathbf{k}}_t(x), \mathbf{k}_t^{time}$ are defined in Sec. 4.2. In the above formulation, we first select a batch of B categorical choices $h_t^b, \forall b \leq B$ using TV.EXP3.M. Then, we utilize the kernel in Eqn. (5) to learn the time-varying GP model to select the next batch of continuous points $x_t^b, \forall b \leq B$ following Eqn. (3), as in PB2-Rand. Each agent b will then train with $[x_t^b, h_t^b]$. We do not need to set the parameter λ as it can be optimized alongside the other kernel parameters (see the Appendix, Sec: D.3). Thus, PB2-Mix in principle achieves the best of both worlds: it is able to learn the relative importance of the continuous and categorical kernels, while including all of the data.

5.3 Main theoretical results

We extend the sublinear regret bound from PB2-R to PB2-D, providing performance guarantees. Under the categorical-continuous setting, we show the equivalence that minimizing the cumulative regret R_T is equivalent to maximizing the reward, presented in Lem. 4 in the appendix. Our main theoretical result is to derive the regret bound for PB2-D.

Theorem 2. *The PB2-D algorithm has the cumulative regret bounded with high probability*

$$\mathbb{E}\left[R_{TB}\right] \lesssim \mathcal{O}\left(V\sqrt{\frac{CT}{B}\ln T} + \sqrt{\frac{T_{c_t^*}^2}{\tilde{N}B}(\gamma_{\tilde{N}B} + \left[\tilde{N}B\right]^3\omega)}\right)$$

for $\tilde{N} \to T_{c_t^*}$, $\omega \to 0$, *and* $V \ll \sqrt{T}$.

From Theorem 1, we have $T_{c_t^*} \to \infty$ when $T \to \infty$. Asymptotically PB2-D achieves sublinear convergence rate, i.e., $\lim_{T\to\infty} \frac{\mathbb{E}[R_{TB}]}{T} = 0$. We refer to Sec. D.2 in the Appendix for proofs.

6 Experiments

Our experiments focus two settings. First, we seek to evaluate the efficacy of our approaches in optimizing continuous and categorical hyperparameters in a synthetic task where we know the variables are highly dependent. Secondly, we test our algorithms in the AutoRL setting, applying them to automatic data augmentation for challenging procedurally generated environments [48].

6.1 Synthetic Function with Dependency

We begin with a simple synthetic function with one continuous and one categorical variable. The categorical variable has two options $h_t^b \sim \{\sin, \cos\}$ and the continuous variable has bounds $x_t^b \sim [0, \frac{\pi}{2}]$. The blackbox function $f : [x, h] \to \mathbb{R}$ returns $h(x)$, and the regret is equal to $1 - f_t(.)$. It is clear to see the function is maximized (and regret minimized) at $\{\sin, \frac{\pi}{2}\}$ and $\{\cos, 0\}$. We compare the three new PB2 variants (PB2-I, PB2-D and PB2-C2) against PB2-R, PBT and Random search. At each iteration, the worst performing agent is replaced by the best and the categories and continuous variables are re-selected.

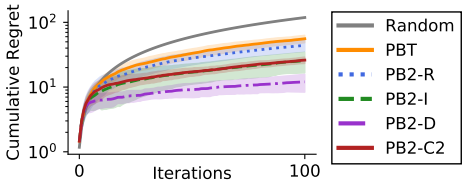


Figure 2: Mean $\pm$1std cumulative regret for 20 runs.

We run the example 20 times, each time with a population size of 4 agents. The results in Figure 2 show the mean cumulative regret, with standard deviation shaded. As we see, random search has linear regret, while PBT and PB2-R both provide incremental improvements. Our new approaches outperform, led by PB2-D. Thus, we see that when there is strong dependence between the two hyperparameters, the methods which explicitly model this significantly outperform. For details, see the following notebook `http://bit.ly/synthetic_func`, including open source implementations of all the discussed algorithms.

6.2 Learning Hyperparameters and Data Augmentation for Generalization in RL

There has recently been increased interest in testing RL algorithms procedurally generated environments [40, 14]. These typically consist of a set of parameters which are used to generate *levels*, consisting of changes to the observation or layout of the environment, requiring agents to generalize [29, 52]. In this paper we focus on the Procgen environment [14]. Given the recent success of data augmentation in Procgen, we adopt the author's implementation of DrAC [48]. As well as the categorical data augmentation, we also tune the regularization coefficient α_r, the PPO clip parameter ϵ, the learning rate and entropy exploration coefficient. The bounds for each, as well as the fixed hyperparameters, are given in the Appendix (Section A, Table A.1 and 3). We learn these hyperparameters, as well as the categorical data augmentation *on the fly* in a single training run, starting with random samples from a wide range.

We train a population of $B = 4$ agents, each for 25M steps, with $t_{\text{ready}} = 500$k steps, thus the explore step is called approximately fifty times. We use five Procgen games: BigFish, CaveFlyer, CoinRun, Leaper an StarPilot and run each algorithm five times. At the end, each agent is evaluated 100 times on the train and test set and we present the test performance from the agent with the best final training

performance in the population. The results, as well as the PPO and UCB-DrAC baselines, are shown in Table 2. For full train and test curves, see Fig. 7 and 3 in the Appendix.

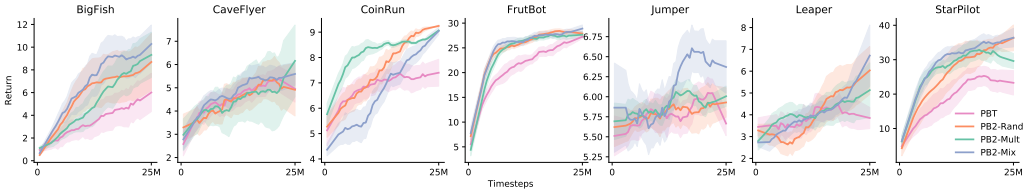


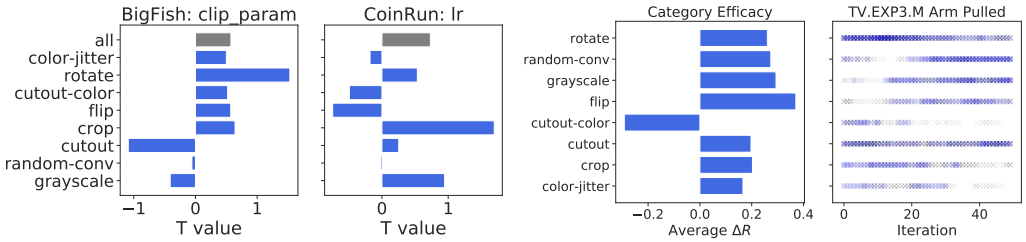
Figure 3: **Test Performance**: Learning curves for all five Procgen games. Plots show the mean $\pm$1std for the average test performance of the top two training agents in the population. Results are averaged over 5 seeds.

Table 2: Test performance for five Procgen games. † indicates training was conducted for a single agent for 25M timesteps. However, hyperparameters were initialized at optimized values, meaning the effective number of timesteps is much higher. Final performance is the average of 100 trials and results were taken from [48]. ‡ indicates training was conducted by a population of four agents for 25M timesteps each, with several hyperparameters initialized at random. Each agent is evaluated for 100 trials on train and test levels, and the performance is presented is the mean test performance of the agent with the best training performance.

Environment	PPO†	UCB-DrAC†	PBT‡	PB2-Rand‡	PB2-Mult‡	PB2-Mix‡
BigFish	4.0 ± 1.2	9.7 ± 1.0	x	x	x	x
CaveFlyer	5.1 ± 0.9	5.3 ± 0.9	x	x	x	x
CoinRun	$\mathbf{8.5 \pm 0.5}$	$\mathbf{8.5 \pm 0.5}$	x	x	x	x
FruitBot	26.7 ± 0.8	28.3 ± 0.9	x	x	x	x
Jumper	5.8 ± 0.5	6.4 ± 0.6	x	x	x	x
Leaper	4.9 ± 0.7	5.0 ± 0.3	x	x	x	x
StarPilot	24.7 ± 3.4	30.2 ± 2.8	x	x	x	x
Normalized Returns (%)			x	100	x	x

Our main findings are that with only four agents we are able to train policies that match or outperform state-of-the-art methods, which used a much larger grid search. We see that PB2-R significantly outperforms PBT, and furthermore our new approaches modeling categorical variables improve upon PB2-R. Notably, we see the strongest performance from PB2-C2, which is able to exploit the dependence between all variables in a joint kernel.

Do we need to model dependence? To explore the dependence of the parameters in the Procgen setting, we ran linear regression models attempting to predict the change in reward ($f_t(.)$) for each *individual* hyperparameter, *conditioned* on the category currently being used. In Fig. **??** we show the t-stats for two separate examples, with the full grid in the Appendix (Fig. 8, Section B). As we see, the relationship between continuous variables and learning performance varies depending on the category selected, confirming a dependence in Procgen [61]. We believe this led to the outperformance for PB2-C2, while PB2-D lacked sufficient data with the batch size smaller than the number of categories, which is problematic as it must maintain an individual model for each category.



(a) (b)

Figure 4: (a) T-statistics for univariate regressions predicting reward change given a continuous hyperparameter, conditioned on category. (b) Left: Mean reward change for each category in the BigFish environment, across all experiments. Right: All arms selected by TV.EXP3.M, for PB2-I, PB2-D and PB2-C2.

We also investigate the effectiveness of TV.EXP3.M in selecting the data augmentation. In Fig. 4 we show the mean change in reward for each category in the BigFish environment, as well as all the categories selected by the three new PB2 variants using TV.EXP3.M. As we see, after

initial exploration, the algorithm avoids the cutout-color augmentation which is by far the weakest performing.

Finally, we consider an ablation study where we use `TV-EXP3.M` to select the categories, but ignore this information and select the continuous variables as in PB2-Rand. Essentially this is a stepping stone between PB2-Rand and our new algorithms PB2-Mult and PB2-Mix. In Fig XXX we show

Does it scale? PBT algorithms typically make use of large population sizes, for example more than 20 agents. Thus far, we have only required four agents to achieve strong performance on the Procgen tasks considered, but the question remains: how do these new methods scale? Our hypothesis would be that PB2-Mult improves with more data, since it filters the dataset for each category, while we would suspect PB2-Mix to continue outperformance over PB2-Rand. We test this hypothesis on the BigFish task, training a population size of 12 for PB2-Rand, PB2-Mult and PB2-Mix, reporting the mean test performance from three seeds in Fig. 5. As we

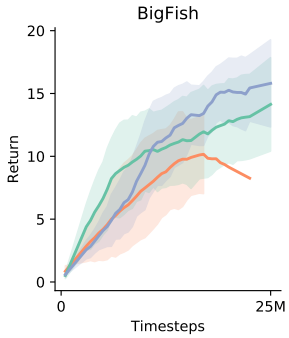


Figure 5: Mean test performance, standard deviation shaded (3 seeds).

6.3 Discussion

Our results make a strong case for population based AutoRL methods in the Procgen benchmark, and we believe they may also perform well on other procedurally generated environments. However, we note there are some limitations in our approach. Firstly, PB2-Mult, which has the strongest theoretical results and performance on the synthetic task, was less impressive in Procgen. This may be due to the continuous/categorical dependence not being strong enough to justify only using a small subset of the data when exploring. We believe PB2-Mix is possibly the best bet for a novel problem, as it has the ability to trade off between the continuous and categorical kernels. However, optimizing this parameter at each explore step does introduce additional computational complexity, which increases the runtime, which will scale with the amount of data. We note that for Procgen, this gain in runtime is negligible, as the training process is up to 24 hours per agent.

Finally we note an addition weakness, and potential area for improvement in our approach: we make decisions in the explore/exploit step based on *training* performance (on changing levels), ignoring generalization ability. This may lead to accidentally removing the best agents on the test set, while it may also lead to inaccuracy in our GP models. One potentially promising approach may be to consider a fixed set of "validation levels", learned using techniques such as *Prioritized Level Replay* [28].

7 Conclusion and Future Work

In this paper we expand the capabilities of population based bandits (PB2) by making it possible to efficiently select *both* continuous and categorical hyperparameters. We introduced a new time varying multi-armed bandit algorithm for selecting categorical hyperparameters, and present a hierarchical approach to subsequently select continuous parameters. We believe this work is an important step in increasing the capability of population based approaches for AutoRL. This should provide benefits at both ends of the spectrum: it can make RL accessible to a wider audience by significantly lowering the cost of tuning hyperparameters, while also potentially leading to stronger performance for those with larger resources. We also showed the effectiveness of learning both hyperparameters and data augmentation on the fly, for challenging procedurally generated environments. We are not the first to propose population based methods in this setting [51], but we provided strong evidence that PBT-style hyperparameter tuning can provide significant benefits.

We are particularly excited by the many future directions from here. The most natural is to learn more: can we learn policy architectures in the explore step, for example making use of kernels between networks [68]? Can we learn algorithms altogether, extending exciting recent work [41, 13]? For the specific case of Procgen environments, can we jointly learn hyperparameters, augmentation and prioritize which levels to train on [28]? In addition, since we have a population of agents, can we improve performance by sharing experience across agents [19]? Can we benefit from encouraging

these agents to be diverse with respect to one another [45, 47]? We are also curious to see whether recent innovations in other areas of AutoML can be combined in a PBT-style framework [50]. We think the combination of these ideas could lead to dramatic gains in performance for challenging RL environments.

References

[1] A. Alvi, B. Ru, J.-P. Calliess, S. Roberts, and M. A. Osborne. Asynchronous batch Bayesian optimisation with improved local penalisation. In *Proceedings of the 36th International Conference on Machine Learning*, pages 253–262, 2019.

[2] M. Andrychowicz, A. Raichuk, P. Stańczyk, M. Orsini, S. Girgin, R. Marinier, L. Hussenot, M. Geist, O. Pietquin, M. Michalski, S. Gelly, and O. Bachem. What matters for on-policy deep actor-critic methods? a large-scale study. In *International Conference on Learning Representations*, 2021.

[3] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire. The nonstochastic multiarmed bandit problem. *SIAM journal on computing*, 32(1):48–77, 2002.

[4] A. G. Barto and R. S. Sutton. Goal seeking components for adaptive intelligence: An initial assessment. Technical report, MASSACHUSETTS UNIV AMHERST DEPT OF COMPUTER AND INFORMATION SCIENCE, 1981.

[5] R. Bellman. *Dynamic Programming*. Dover Publications, 1957.

[6] J. Bergstra, D. Yamins, and D. Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *Proceedings of the 30th International Conference on Machine Learning*, 2013.

[7] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. P. de Oliveira Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, and S. Zhang. Dota 2 with large scale deep reinforcement learning. *CoRR*, abs/1912.06680, 2019.

[8] I. Bogunovic, J. Scarlett, and V. Cevher. Time-Varying Gaussian Process Bandit Optimization. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, 2016.

[9] E. Brochu, V. M. Cora, and N. de Freitas. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *CoRR*, abs/1012.2599, 2010.

[10] J. S. O. Ceron and P. S. Castro. Revisiting rainbow: Promoting more insightful and inclusive deep reinforcement learning research. In *Deep Reinforcement Learning Workshop, NeurIPS 2020*, 2020.

[11] Y. Chen, A. Huang, Z. Wang, I. Antonoglou, J. Schrittwieser, D. Silver, and N. de Freitas. Bayesian optimization in AlphaGo. *CoRR*, abs/1812.06855, 2018.

[12] H. L. Chiang, A. Faust, M. Fiser, and A. G. Francis. Learning navigation behaviors end-to-end with autorl. *IEEE Robotics Autom. Lett.*, 4(2):2007–2014, 2019.

[13] J. D. Co-Reyes, Y. Miao, D. Peng, Q. V. Le, S. Levine, H. Lee, and A. Faust. Evolving reinforcement learning algorithms. In *International Conference on Learning Representations*, 2021.

[14] K. Cobbe, C. Hesse, J. Hilton, and J. Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *Proceedings of the 37th International Conference on Machine Learning*, pages 2048–2056, 2020.

[15] K. Cobbe, O. Klimov, C. Hesse, T. Kim, and J. Schulman. Quantifying generalization in reinforcement learning. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, Proceedings of Machine Learning Research. PMLR, 2019.

[16] T. Desautels, A. Krause, and J. W. Burdick. Parallelizing exploration-exploitation tradeoffs in Gaussian Process bandit optimization. *Journal of Machine Learning Research*, 15(119):4053–4103, 2014.

[17] L. Engstrom, A. Ilyas, S. Santurkar, D. Tsipras, F. Janoos, L. Rudolph, and A. Madry. Implementation matters in deep RL: A case study on PPO and TRPO. In *International Conference on Learning Representations*, 2020.

[18] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, S. Legg, and K. Kavukcuoglu. IMPALA: Scalable distributed deep-RL with importance weighted actor-learner architectures. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

[19] J. K. Franke, G. Koehler, A. Biedenkapp, and F. Hutter. Sample-efficient automated deep reinforcement learning. In *International Conference on Learning Representations*, 2021.

[20] R. Gandhi, S. Khuller, S. Parthasarathy, and A. Srinivasan. Dependent rounding and its applications to approximation algorithms. *Journal of the ACM (JACM)*, 53(3):324–360, 2006.

[21] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger. Deep reinforcement learning that matters. *AAAI*, 2018.

[22] P. Hennig and C. J. Schuler. Entropy search for information-efficient global optimization. *J. Mach. Learn. Res.*, 13(null):1809–1837, June 2012.

[23] D. Ho, E. Liang, X. Chen, I. Stoica, and P. Abbeel. Population based augmentation: Efficient learning of augmentation policy schedules. In *Proceedings of the 36th International Conference on Machine Learning*, pages 2731–2741. 2019.

[24] F. Hutter, L. Kotthoff, and J. Vanschoren, editors. *Automated Machine Learning: Methods, Systems, Challenges*. Springer, 2018. In press, available at http://automl.org/book.

[25] R. Islam, P. Henderson, M. Gomrokchi, and D. Precup. Reproducibility of benchmarked deep reinforcement learning tasks for continuous control. *CoRR*, abs/1708.04133, 2017.

[26] M. Jaderberg, W. M. Czarnecki, I. Dunning, L. Marris, G. Lever, A. G. Castañeda, C. Beattie, N. C. Rabinowitz, A. S. Morcos, A. Ruderman, N. Sonnerat, T. Green, L. Deason, J. Z. Leibo, D. Silver, D. Hassabis, K. Kavukcuoglu, and T. Graepel. Human-level performance in 3d multiplayer games with population-based reinforcement learning. *Science*, 364(6443):859–865, 2019.

[27] M. Jaderberg, V. Dalibard, S. Osindero, W. M. Czarnecki, J. Donahue, A. Razavi, O. Vinyals, T. Green, I. Dunning, K. Simonyan, C. Fernando, and K. Kavukcuoglu. Population based training of neural networks. *CoRR*, abs/1711.09846, 2017.

[28] M. Jiang, E. Grefenstette, and T. Rocktäschel. Prioritized level replay. In *The International Conference on Machine Learning*. 2021.

[29] N. Justesen, R. R. Torrado, P. Bontrager, A. Khalifa, J. Togelius, and S. Risi. Procedural level generation improves generality of deep reinforcement learning. *NeurIPS Deep Reinforcement Learning Workshop*, 2018.

[30] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, and S. Levine. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *CoRR*, abs/1806.10293, 2018.

[31] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[32] I. Kostrikov, D. Yarats, and R. Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. In *International Conference on Learning Representations*. 2021.

[33] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*. 2012.

[34] M. Laskin, K. Lee, A. Stooke, L. Pinto, P. Abbeel, and A. Srinivas. Reinforcement learning with augmented data. In *Advances in Neural Information Processing Systems 33*. 2020.

[35] M. Laskin, A. Srinivas, and P. Abbeel. CURL: Contrastive unsupervised representations for reinforcement learning. In *Proceedings of the 37th International Conference on Machine Learning*, 2020.

[36] A. Li, O. Spyra, S. Perel, V. Dalibard, M. Jaderberg, C. Gu, D. Budden, T. Harley, and P. Gupta. A generalized framework for population based training. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery Data Mining*, 2019.

[37] S. Liu, G. Lever, N. Heess, J. Merel, S. Tunyasuvunakool, and T. Graepel. Emergent coordination through competition. In *International Conference on Learning Representations*, 2019.

[38] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller. Playing Atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[39] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, W. Paul, M. I. Jordan, and I. Stoica. Ray: A distributed framework for emerging AI applications. *CoRR*, abs/1712.05889, 2017.

[40] A. Nichol, V. Pfau, C. Hesse, O. Klimov, and J. Schulman. Gotta learn fast: A new benchmark for generalization in RL. *CoRR*, abs/1804.03720, 2018.

[41] J. Oh, M. Hessel, W. M. Czarnecki, Z. Xu, H. van Hasselt, S. Singh, and D. Silver. Discovering reinforcement learning algorithms. In *Advances in Neural Information Processing Systems 33*. 2020.

[42] OpenAI, M. Andrychowicz, B. Baker, M. Chociej, R. Józefowicz, B. McGrew, J. W. Pachocki, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba. Learning dexterous in-hand manipulation. *CoRR*, abs/1808.00177, 2018.

[43] C. Packer, K. Gao, J. Kos, P. Krahenbuhl, V. Koltun, and D. Song. Assessing generalization in deep reinforcement learning. *CoRR*, 2019.

[44] J. Parker-Holder, V. Nguyen, and S. Roberts. Provably efficient online hyperparameter optimization with population-based bandits. In *Advances in Neural Information Processing Systems*. 2020.

[45] J. Parker-Holder, A. Pacchiano, K. Choromanski, and S. Roberts. Effective diversity in population-based reinforcement learning. In *Advances in Neural Information Processing Systems 33*. 2020.

[46] S. Paul, V. Kurin, and S. Whiteson. Fast efficient hyperparameter tuning for policy gradients. *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[47] J. K. Pugh, L. B. Soros, and K. O. Stanley. Quality diversity: A new frontier for evolutionary computation. *Frontiers in Robotics and AI*, 3:40, 2016.

[48] R. Raileanu, M. Goldstein, D. Yarats, I. Kostrikov, and R. Fergus. Automatic data augmentation for generalization in deep reinforcement learning. *CoRR*, abs/2006.12862, 2020.

[49] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[50] E. Real, C. Liang, D. R. So, and Q. V. Le. Automl-zero: Evolving machine learning algorithms from scratch. In *Proceedings of the 37th International Conference on Machine Learning*. 2020.

[51] S. Risi and J. Togelius. Neuroevolution in games: State of the art and open challenges. *IEEE Transactions on Computational Intelligence and AI in Games*, 2017.

[52] S. Risi and J. Togelius. Procedural content generation: From automatically generating game levels to increasing generality in machine learning. *Nature*, 2020.

[53] B. X. Ru, A. S. Alvi, V. Nguyen, M. A. Osborne, and S. J. Roberts. Bayesian optimisation over multiple continuous and categorical inputs. In *Proceedings of the 37th International Conference on Machine Learning*. 2020.

[54] F. Runge, D. Stoll, S. Falkner, and F. Hutter. Learning to design RNA. In *International Conference on Learning Representations*, 2019.

[55] S. Schmitt, J. J. Hudson, A. Zídek, S. Osindero, C. Doersch, W. M. Czarnecki, J. Z. Leibo, H. Küttler, A. Zisserman, K. Simonyan, and S. M. A. Eslami. Kickstarting deep reinforcement learning. *CoRR*, abs/1803.03835, 2018.

[56] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning*, 2015.

[57] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[58] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *CoRR*, 2017.

[59] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. P. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529:484–489, 2016.

[60] J. Snoek, H. Larochelle, and R. P. Adams. Practical Bayesian optimization of machine learning algorithms. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*. 2012.

[61] X. Song, Y. Du, and J. Jackson. An empirical study on hyperparameters and their interdependence for RL generalization. *ICML 2019 Workshop "Understanding and Improving Generalization in Deep Learning"*, 2019.

[62] X. Song, Y. Jiang, S. Tu, Y. Du, and B. Neyshabur. Observational overfitting in reinforcement learning. In *International Conference on Learning Representations*, 2020.

[63] N. Srinivas, A. Krause, S. Kakade, and M. Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, 2010.

[64] R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[65] C. Szepesvári. *Algorithms for reinforcement learning*. Morgan & Claypool Publishers, 2010.

[66] A. A. Taïga, W. Fedus, M. C. Machado, A. C. Courville, and M. G. Bellemare. Benchmarking bonus-based exploration methods on the arcade learning environment. *CoRR*, abs/1908.02388, 2019.

[67] T. Uchiya, A. Nakamura, and M. Kudo. Algorithms for adversarial bandit problems with multiple plays. In *International Conference on Algorithmic Learning Theory*, pages 375–389. Springer, 2010.

[68] X. Wan, B. Ru, X. Dong, and M. Osborne. Interpretable neural architecture search via Bayesian optimisation with Weisfeiler-Lehman kernels. In *International Conference on Learning Representations*, 2021.

[69] K. Wang, B. Kang, J. Shao, and J. Feng. Improving generalization in reinforcement learning with mixture regularization. In *Advances in Neural Information Processing Systems 33*. 2020.

[70] Z. Xu, H. P. van Hasselt, and D. Silver. Meta-gradient reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 31, pages 2396–2407, 2018.

[71] C. Ye, A. Khalifa, P. Bontrager, and J. Togelius. Rotation, translation, and cropping for zero-shot generalization. In *2020 IEEE Conference on Games (CoG)*, pages 57–64, 2020.

[72] T. Zahavy, Z. Xu, V. Veeriah, M. Hessel, J. Oh, H. van Hasselt, D. Silver, and S. Singh. A self-tuning actor-critic algorithm. In *Advances in Neural Information Processing Systems*, 2020.

[73] B. Zhang, R. Rajan, L. Pineda, N. Lambert, A. Biedenkapp, K. Chua, F. Hutter, and R. Calandra. On the importance of hyperparameter optimization for model-based reinforcement learning. In *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, 2021.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 6.3.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A] The sole aim of this work is to make it possible to use *existing methods* more effectively.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] Theory is presented in Section 5.
   (b) Did you include complete proofs of all theoretical results? [Yes] Please see Section D.2 and D.1 in the Appendix
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Code for the Procgen experiments is attached in the supplementary materials. Code for the synthetic problem is released as a notebook. The full algorithm will be released as part of the ray library once published.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Appendix A.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] All Procgen results include five seeds, with standard deviations in the tables and plots.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Appendix A.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] Cited the Procgen benchmark.
   (b) Did you mention the license of the assets? [Yes] Cited the Procgen benchmark.
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A] Not applicable.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] Not applicable.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] Not applicable.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A] Did not use human subjects.
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A] Did not use human subjects.
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A] Did not use human subjects.

A Implementation Details

A.1 Hyperparameter Ranges

RL Hyperparameters: Below are the continuous hyperparameters optimized as well as the fixed parameters, which were all taken from **(author?)** [48]. For the categorical variables, we use the data augmentations used in [48, 34, 32]: *crop, grayscale, cutout, cutout-color, flip, rotate, random convolution and color-jitter*.

Table 3: DrAC Learned Hyperparameters

Parameter	Value
PPO Clip (ϵ)	$[0.01, 0.5]$
Learning Rate	$[10^{-5}, 10^{-3}]$
Entropy Coeff	$[0, 0.2]$
Regularization Parameter (α_r)	$[0.01, 0.5]$

Table 4: DrAC Fixed Hyperparameters

Parameter	Value
γ	0.999
λ	0.95
# timesteps per rollout	256
# epochs per rollout	3
# minibatches per epoch	8
optimizer	Adam

PB2 Hyperparameters: We use the same UCB hyperparameters as in **(author?)** [44], which come from **(author?)** [8]. Concretely, we set $c1 = 0.2$, $c2 = 0.4$. We believe performance may be increased by selecting these parameters more carefully. However, it is promising to see that this fine tuning is not crucial for performance.

A.2 Environment Details

We used five games from the Procgen environment [14]. We chose these games as there appeared to be a greater degree of variance in the optimal data augmentation in **(author?)** [48]. All protocols for the environment are from **(author?)** [48], as we followed the author's open source implementation for agent training[1], and only varied the hyperparameters and data augmentation chosen.

(a) **BigFish** (b) **CaveFlyer** (c) **CoinRun** (d) **Leaper** (e) **StarPilot**

Figure 6: Example levels from the five games considered from [14]

A.3 Infrastructure Details

Each trial was run a single GPU, taking around three days to train all four agents. This was due to the nature of our existing compute infrastructure and could likely be made significantly faster with parallelization. The existing code is thus not optimized to work in a distributed fashion, however it will be open sourced in a similar structure to the open source PB2-R implementation (part of the Ray library [39]). Since we followed the open source version of PB2-R, it is a trivial extension to include PB2-I, PB2-D and PB2-C2, using new kernels provided both in the supplementaty material (as part of the research code) or the synthetic function notebook (see the link in Section 6).

[1] see: `https://github.com/rraileanu/auto-drac`

B Additional Results

In Table 5 we show training performance for the agents shown in the main paper. Interestingly, while PB2-C2 is the best performing algorithm, we see strong performance for PB2-R, resulting in a large generalization gap. We show learning curves for all algorithms in Fig. 7 & 3. In the plots we show the mean of the top two training agents, to reduce noise.

Table 5: **Train** performance for five Procgen games. † indicates training was conducted for a single agent for 25M timesteps. However, hyperparameters were initialized at optimized values, meaning the effective number of timesteps is much higher. Final performance is the average of 100 trials and results were taken from [48]. $\star$ indicates training was conducted by a population of four agents for 25M timesteps each, with several hyperparameters initialized at random. Each agent is evaluated for 100 trials on train and test levels, and the performance is presented is the agent in the population with the best training reward. The "Normalized" score is taken as the average gain in mean performance (across all five environments) vs. the PPO baseline, as used in the literature.

Regime	Algorithm	BigFish	CaveFlyer	CoinRun	Leaper	StarPilot	Normalized
Single Agent† (Grid Search)	PPO	8.9 ± 1.5	6.8 ± 0.6	9.3 ± 0.3	5.5 ± 0.4	29.8 ± 2.3	100
	UCB-DrAC	13.2 ± 2.2	5.7 ± 0.6	9.5 ± 0.3	5.3 ± 0.5	35.3 ± 2.2	109.8
Population$\star$ (Random Cats)	PBT	13.0 ± 3.3	$\mathbf{7.0 \pm 0.4}$	8.2 ± 0.8	4.1 ± 0.5	33.6 ± 7.4	104.9
	PB2-R	$\mathbf{16.8 \pm 5.4}$	6.7 ± 1.7	$\mathbf{10.0 \pm 0.0}$	$\mathbf{6.6 \pm 2.1}$	$\mathbf{42.5 \pm 2.8}$	**131.5**
Population$\star$ (TV-EXP3.M)	PB2-I	$\mathbf{16.8 \pm 2.0}$	6.7 ± 0.6	$\mathbf{9.9 \pm 0.1}$	6.1 ± 2.1	$\mathbf{43.4 \pm 3.0}$	130.1
	PB2-D	16.6 ± 3.7	$\mathbf{7.0 \pm 1.6}$	$\mathbf{9.9 \pm 0.2}$	5.1 ± 2.1	40.7 ± 1.3	125.0
	PB2-C2	16.3 ± 3.0	$\mathbf{7.3 \pm 0.9}$	9.6 ± 0.2	$\mathbf{7.1 \pm 2.3}$	41.8 ± 2.7	**132.6**

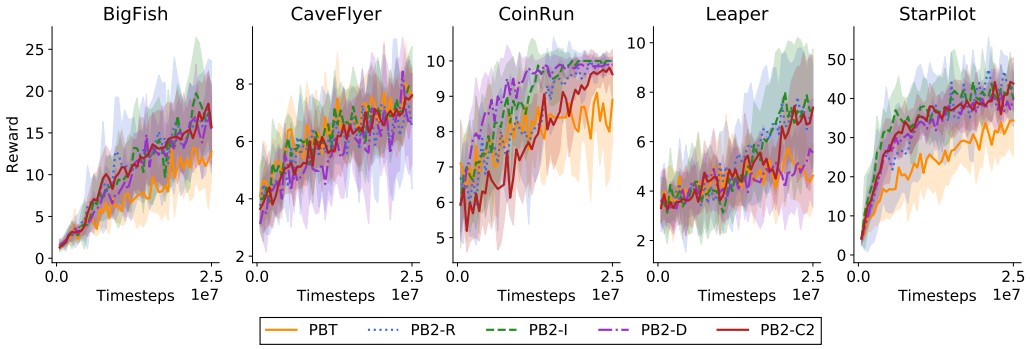


Figure 7: **Train Performance**: Learning curves for all five Procgen games. Plots show the mean $\pm$1std for the best training performance within the population. Results are averaged over 5 seeds.

Is there dependence between data augmentation and hyperparameters in Procgen? Next we consider the dependence between the hyperparameters. For each environment, we fit a linear regression model predicting the change in reward for 500k steps of training (i.e. one step of PB2), with the independent variable being **one** of the continuous hyperparameters. Blue corresponds to the setting where we condition on an individual category (by creating a subset), while grey corresponds to using the entire dataset. We use the data from all trials and show the results in In Fig. 8.

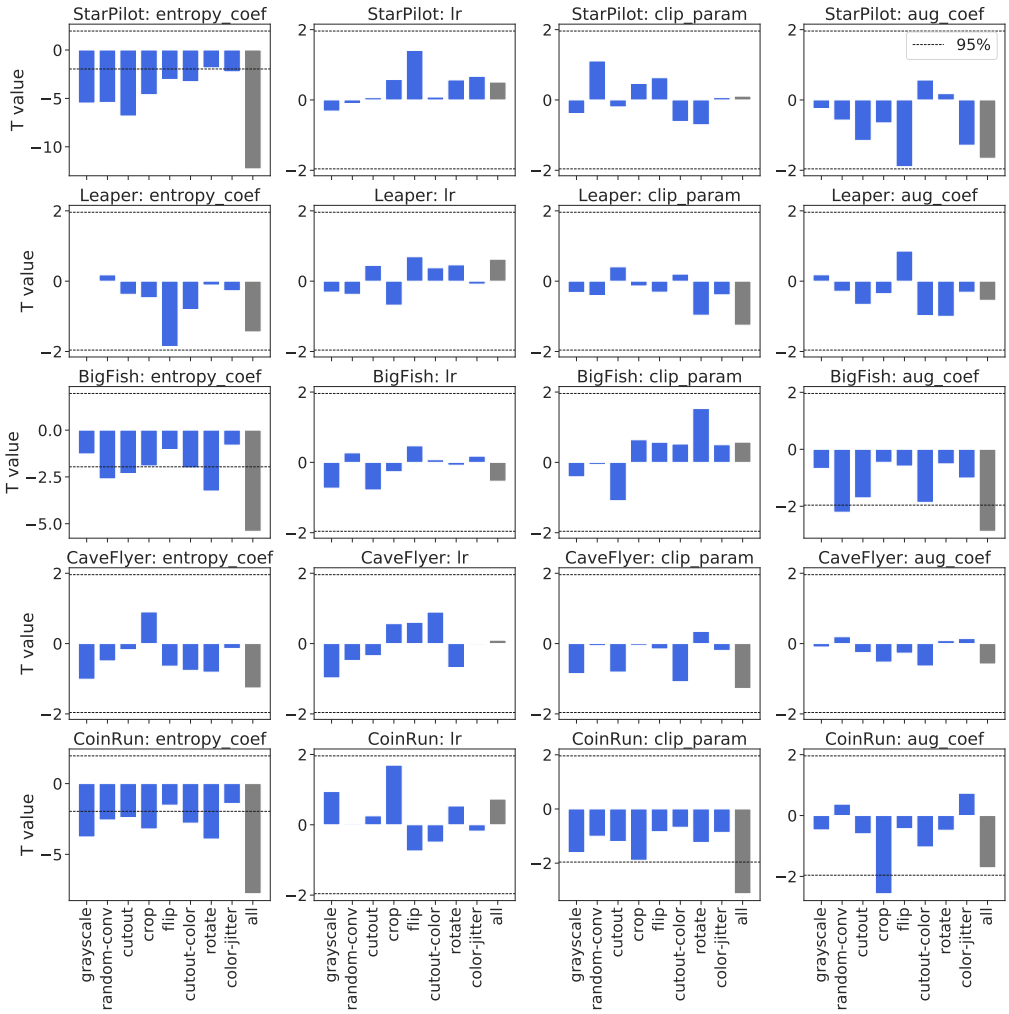


Figure 8: t-statistic for the independent variable (one continuous hyperparameter, in the title of each plot) in predicting the change in reward for the given environment. Blue columns represent the data being from one. The dotted line indicates $p < 0.05$.

As we see, in some cases the category does not have any impact on the relationship, for example the entropy coefficient for CoinRun: an increase in entropy appears to be negatively related with training performance across all augmentations. However, for some settings such as the clip parameter in BigFish, we see large swings in the t-stat depending on category. We even see some settings where the aggregate relationship (grey) is very small, but conditioning on the category makes it worthwhile to tune the hyperparameter, for example the learning rate for CaveFlyer. In this case it may be challenging for PB2-R and PB2-I to achieve gains from tuning the learning rate. We note this analysis is purely illustrative and has many assumptions, in particular 1) we use a linear univariate model 2) we ignore the time varying component.

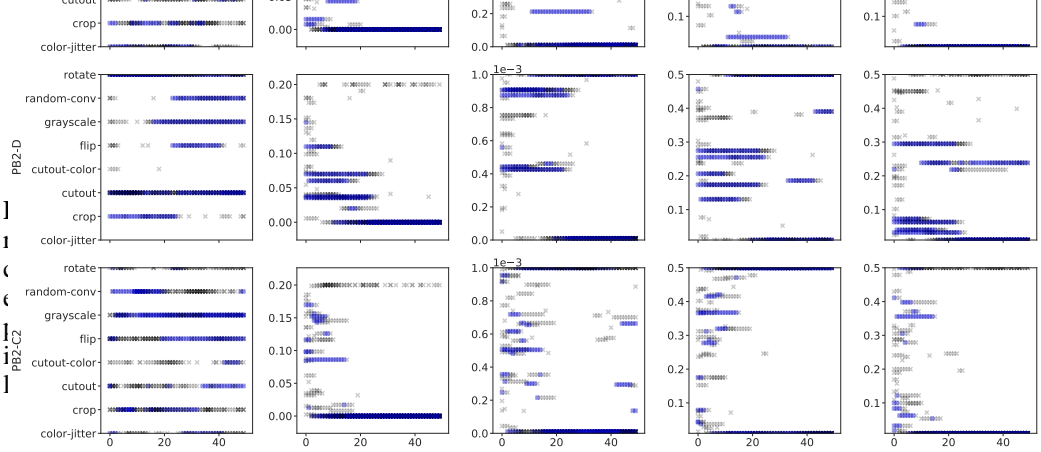


Figure 9: Learned schedules for BigFish by algorithm, for all seeds. Each point corresponds to one agent, blue corresponds to the agent that led to the final best agent.

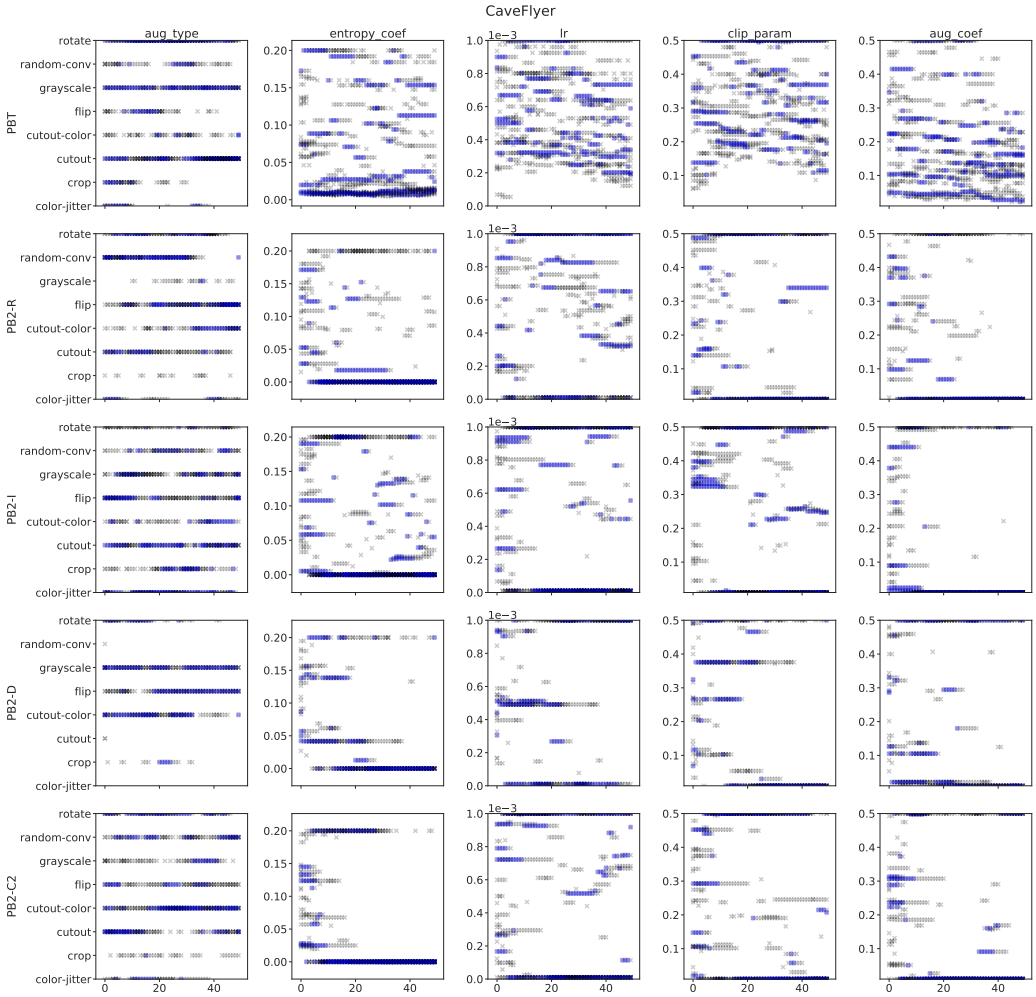


Figure 10: Learned schedules for CaveFlyer by algorithm, for all seeds. Each point corresponds to one agent, blue corresponds to the agent that led to the final best agent.

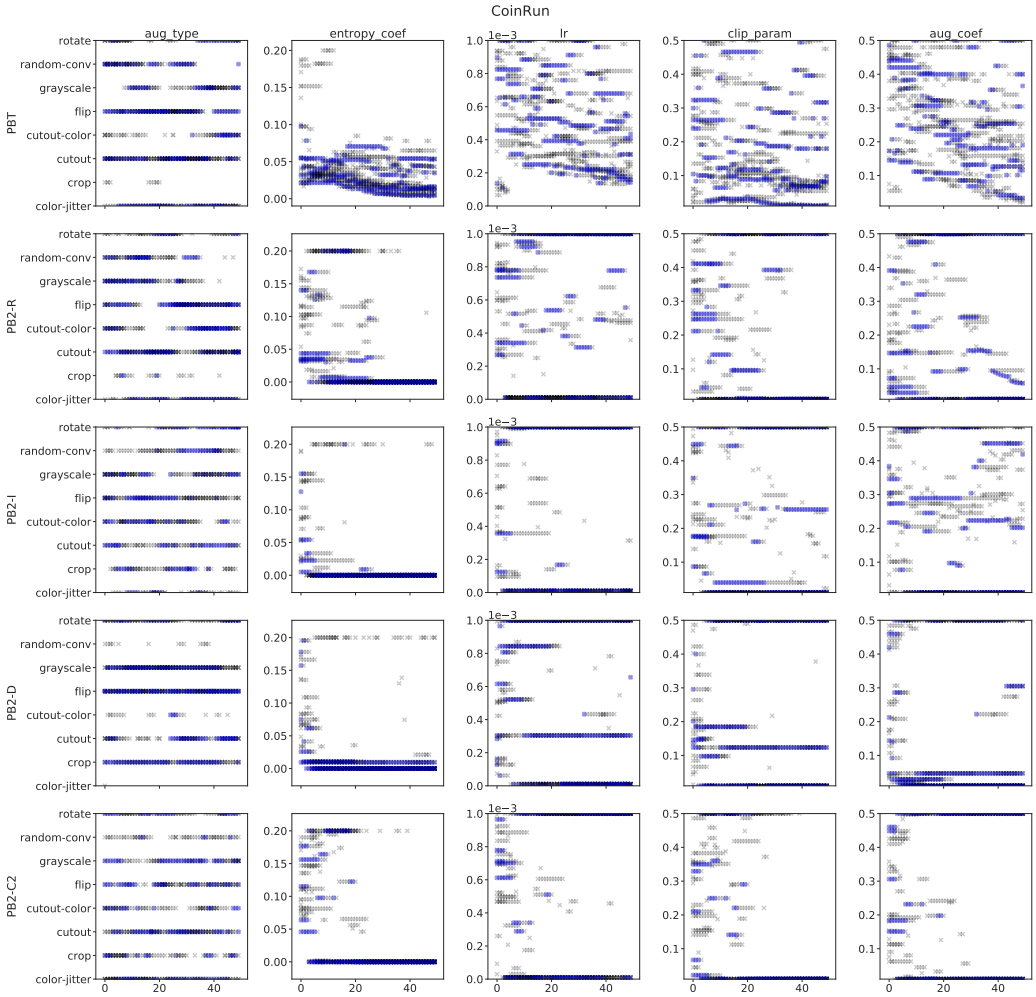


Figure 11: Learned schedules for CoinRun by algorithm, for all seeds. Each point corresponds to one agent, blue corresponds to the agent that led to the final best agent.

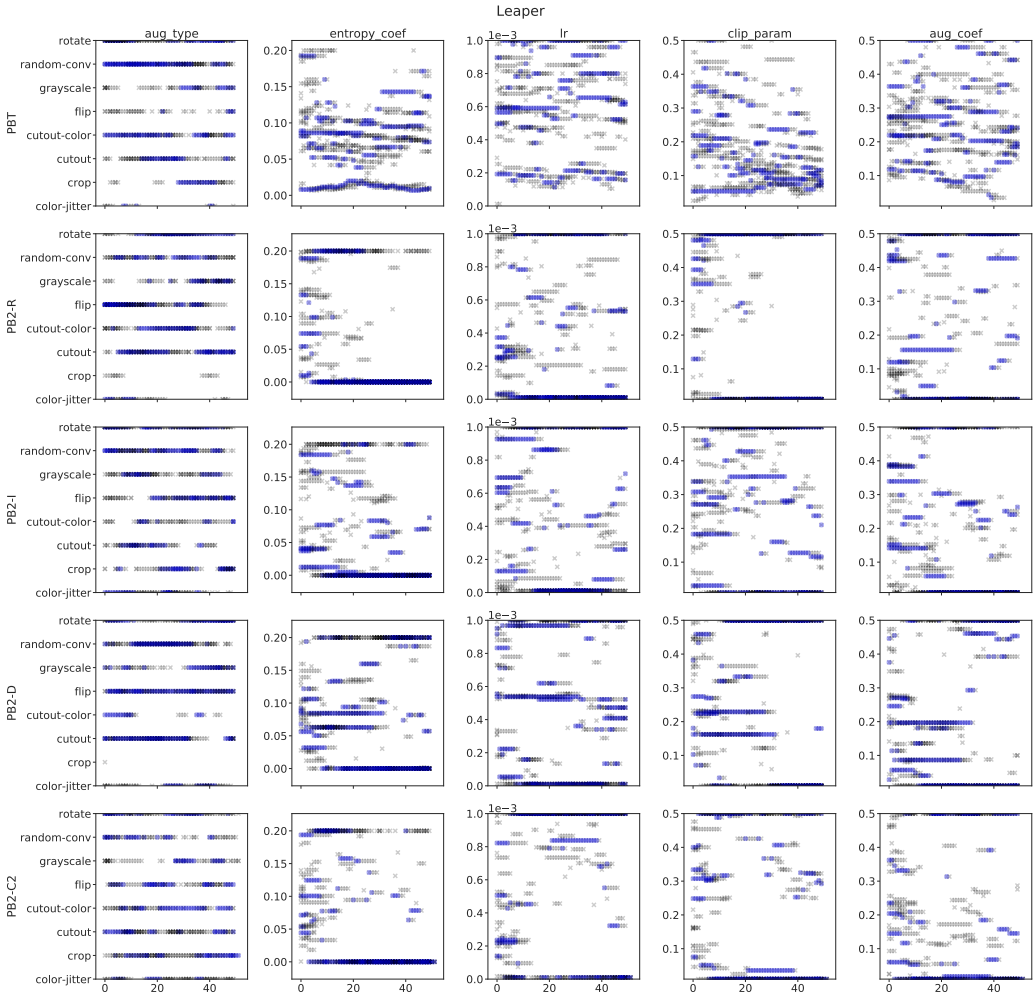


Figure 12: Learned schedules for Leaper by algorithm, for all seeds. Each point corresponds to one agent, blue corresponds to the agent that led to the final best agent.

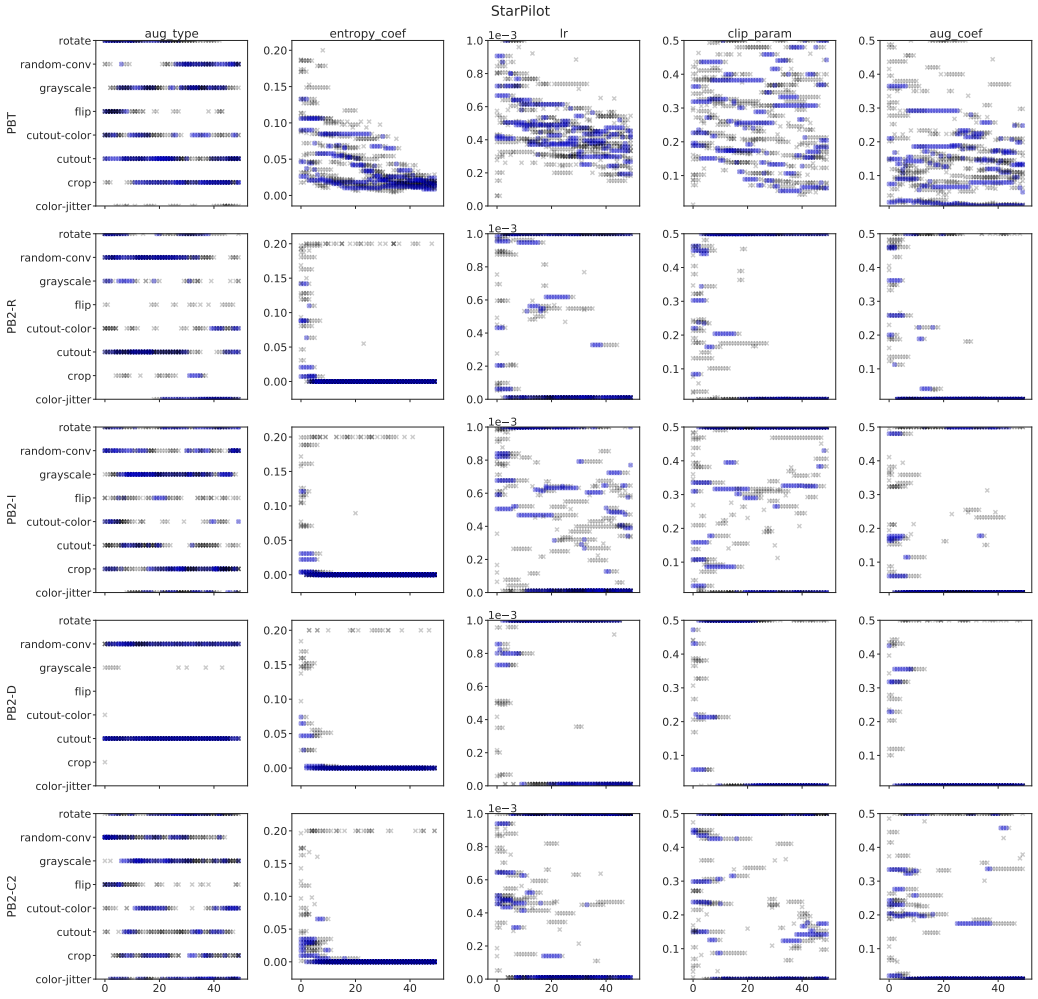


Figure 13: Learned schedules for StarPilot by algorithm, for all seeds. Each point corresponds to one agent, blue corresponds to the agent that led to the final best agent.

C Additional Background

C.1 DrAC

In recent times, there has been increased interest in *generalization* of RL agents, after multiple works showed many RL agents are simply overfitting to a single deterministic training environment [43, 15]. One approach to produce more generalizable agents is data augmentation, which has recently shown impressive results [71, 32, 34, 35]. To formalize the data augmentation step, we follow [32] and define an optimality-invariant state transformation $f : \mathcal{S} \times \mathcal{H} \rightarrow \mathcal{S}$ as a mapping that preserves both the policy π and value function V, i.e. $V(s) = V(f(s, \nu))$ and $\pi(a|s) = \pi(a|f(s, \nu)), \forall s \in \mathcal{S}, \nu \in \mathcal{H}$, where ν are parameters of $f(.)$ drawn from the set of possible parameters $\mathcal{H}$. In this work we focus on the formulation from [48] who propose additional loss terms for regularizing the policy and value function:

$$G_\pi = \mathrm{KL}[\pi_\theta(a|f(s, \nu])|\pi(a|s)], \tag{6}$$

$$G_V = (V_\theta(f(s, \nu)) - V(s))^2 \tag{7}$$

Combining the PPO objective with G_π and G_V produces the *data-regularized actor critic* or DrAC objective as follows:

$$\mathcal{L}_{\mathrm{DrAC}}(\theta) = \mathcal{L}_{\mathrm{PPO}}(\theta) - \alpha_r(G_\pi + G_V) \tag{8}$$

where α_r is the weight of the regularization term, another hyperparameter to consider. The results in [48] show that learning which data augmentation function f to use (from a fixed set) and grid searching over the α_r parameter can achieve new state-of-the-art results in challenging Procgen environments.

Table 6: Notations used in the theoretical analysis.

variable	domain	meaning
C	$\mathcal{N}$	number of categories (number of arms)
T	$\mathcal{N}$	maximum number of bandit update (the number of t_{ready} in PB2)
V	$\mathcal{N}$	number of time segments, how many times the function has been shifted
B	$\mathcal{N}$	batch size (number of parallel agents)
S_t	$list$	a list of B selected categories $[c_{t,1}, c_{t,2}, ..., c_{t,B}]$
e	$\mathcal{R}^+$	this is an Euler's number 2.71828
$\left[p_t^1, ...p_t^C\right]$	$list$	probability vector at iteration t and category $c = 1....C$
w_c	$\mathcal{R}$	weight for categorical c (or arm c)
$W_t = \sum_{c=1}^{C} w_c$	$\mathcal{R}^+$	sum of the weight vector at iteration t
$c_{t,b} \in \{1...C\}$	$\mathcal{N}$	a selection at iteration t at agent b
$c_t = [c_{t,1}, ..., c_{t,B}]$	$list$	a batch of B categorical selections at iteration t
$c_t^* \in \{1...C\}$	$list$	an optimal selection at iteration t
A_v^*	$list$	a list of B elements taking the (same) optimal selection c_t^* at iteration/segment v
$g_t(c)$	$[0, 1]$	a gain (reward) occurred at iteration t by pulling an arm c
$\hat{g}_t(c) = \frac{g_t(c)}{p_t^c + \gamma} \mathbb{I}(c = h_t)$	$\mathcal{R}^+$	a normalized gain
$\alpha = \frac{1}{T}, \eta = 2\gamma = \sqrt{\frac{2 \ln C}{CT}}$	$\mathcal{R}^+$	hyperparameters, set by Theorem 1
$\gamma = \sqrt{\frac{\ln C}{2CT}} \in [0, 1]$	$\mathcal{R}^+$	is the exploration parameter
$G_T(c) = \sum_{t=1}^{T} g_t(c)$	$\mathcal{R}^+$	a total gain if we select an arm c entirely

D Theoretical Results

We derive the theoretical proofs presented in the main paper.

D.1 Time-varying EXP3 Multiple play (TV.EXP3.M)

We present a new algorithm for parallel (or multiple play) multi-armed bandits in the time-varying setting with adversarial feedback. Particularly, we extend the multiple play EXP3 algorithm (or EXP3.M) [67], to the time-varying setting where the unknown reward distribution of each arm can change arbitrarily, but the total number of change points is no more than V times. We refer to Table 6 for the notations used in the proofs.

We summarize the TV.EXP3.M in Algorithm 5 – this is complementary to the brief Algorithm 2 in the main paper. In Algorithm 5, we maintain a set of probability vectors (step 9) for each arm which will be specified and weighted in the optimal way derived by the theory. Then, at each round we select a batch of B arms for parallel evaluations (step 10), observe the reward (step 11) and update the model (step 12,13,14). The normalization steps 3-7 are to prevent from being biased toward a single arm which performs overwhelmingly well, thus overly exploiting.

Theorem 3. *(Theorem 1 in the main paper) Let $T > 0$, $C > 0$, set $\alpha = \frac{1}{T}$ and $\gamma = \min\left\{1, \sqrt{\frac{C \ln(C/B)}{(e-1)BT}}\right\}$, we assume the reward distributions to change at arbitrary time instants, but the total number of change points is no more than V times. The expected regret gained by*

TV.EXP3.M *in a batch satisfies the following sublinear regret bound*

$$\mathbb{E}\left[R_{TB}\right] \leq \left[1+e+V\right]\sqrt{(e-1)\frac{CT}{B}\ln\frac{CT}{B}}.$$

Proof. We refer to Table 6 for notations. We follow the proof technique presented in [3] and [67] to derive the regret bound. Let $W_t = \sum_{c=1}^{C}\omega_t^c$ and using step 9 in Algorithm 5, we have $\frac{\frac{p_t^c}{B}-\frac{\gamma}{C}}{1-\gamma} = \frac{\omega_t^c}{W_t}$. This equation will be used below.

$$\begin{aligned}
\frac{W_{t+1}}{W_t} &= \frac{\sum_{c=1}^{C}\omega_{t+1}^c}{W_t} = \frac{\sum_{c\notin S_t(0)}\omega_{t+1}^c}{W_t} + \frac{\sum_{c\in S_t(0)}\omega_{t+1}^c}{W_t} + \frac{\sum_{c=1}^{C}\frac{e\alpha W_t}{C}}{W_t} \\
&= \sum_{c\notin S_t(0)}\frac{\omega_t^c}{W_t}\times\exp\left(\frac{B\gamma}{C}\hat{g}_t(c)\right) + \frac{\sum_{c\in S_t(0)}\omega_t^c}{W_t} + e\alpha \\
&\leq e\alpha + \sum_{c\notin S_t(0)}\frac{\omega_t^c}{W_t}\left[1+\frac{B\gamma}{C}\hat{g}_t(c)+(e-2)\left(\frac{B\gamma}{C}\right)^2\hat{g}_t^2(c)\right] + \frac{\sum_{c\in S_t(0)}\omega_{t+1}^c}{W_t} \quad \text{by } e^x \leq 1+x+(e-2)x^2 \\
&= e\alpha + \underbrace{\sum_{c\notin S_t(0)}\frac{\omega_t^c}{W_t} + \sum_{c\in S_t(0)}\frac{\omega_{t+1}^c}{W_t}}_{1} + \sum_{c\notin S_t(0)}\frac{\frac{p_t^c}{B}-\frac{\gamma}{C}}{1-\gamma}\left[\frac{B\gamma}{C}\hat{g}_t(c)+(e-2)\left(\frac{B\gamma}{C}\right)^2\hat{g}_t^2(c)\right] \\
&\leq e\alpha + 1 + \frac{B\gamma}{C\left(1-\gamma\right)}\sum_{c\notin S_t(0)}\left(\frac{p_t^c}{B}-\frac{\gamma}{C}\right)\hat{g}_t(c) + \frac{(e-2)}{1-\gamma}\left(\frac{B\gamma}{C}\right)^2\sum_{c\notin S_t(0)}\left(\frac{p_t^c}{B}-\frac{\gamma}{C}\right)\hat{g}_t^2(c) \\
&= e\alpha + 1 + \frac{\gamma}{C\left(1-\gamma\right)}\sum_{c\notin S_t(0)}p_t^c\hat{g}_t(c) - \frac{B\gamma^2}{C^2\left(1-\gamma\right)}\sum_{c\notin S_t(0)}\hat{g}_t(c) \\
&+ \frac{(e-2)\gamma^2 B}{\left(1-\gamma\right)C^2}\sum_{c\notin S_t(0)}p_t^c\hat{g}_t^2(c) - \frac{(e-2)\gamma^3 B^2}{\left(1-\gamma\right)C^2}\sum_{c\notin S_t(0)}\hat{g}_t^2(c) \\
&\leq e\alpha + 1 + \frac{\gamma}{C\left(1-\gamma\right)}\sum_{c\notin S_t(0)}p_t^c\hat{g}_t(c) + \frac{(e-2)\gamma^2 B}{\left(1-\gamma\right)C^2}\sum_{c\notin S_t(0)}p_t^c\hat{g}_t^2(c) \\
&\leq e\alpha + 1 + \frac{\gamma}{C\left(1-\gamma\right)}\sum_{c\in S_t - S_t(0)}g_t(c) + \frac{(e-2)\gamma^2 B}{\left(1-\gamma\right)C^2}\sum_{c\notin S_t(0)}\hat{g}_t(c) \quad \text{by } x^2 \leq x, \forall x\in[0,1] \\
\frac{W_{t+1}}{W_t} &\leq e\alpha + 1 + \frac{\gamma}{C\left(1-\gamma\right)}\sum_{c\in S_t - S_t(0)}g_t(c) + \frac{(e-2)\gamma^2 B}{\left(1-\gamma\right)C^2}\sum_{c\in[C]}\hat{g}_t(c). \qquad (9)
\end{aligned}$$

To demonstrate the time-varying property in our bandit problem, we assume the reward distributions change at arbitrary time instants, but the total number of change points is no more than V times. We split the sequence of total decisions T into V segments such that within each segment we have the *time-invariant* reward function.

We can write V segments as $[T_1, ...T_2), [T_2, ..., T_3), [T_V, ..., T_{V+1})$ where T_v indicates the starting index of the v-th segment, T_{v+1} is the ending index of the v-th segment which is also the starting index of the $v+1$-th segment – using the same notation in EXP3.S [3]. Similarly, we can write the optimal sequence $\underbrace{[c^*_{T_1}, ...c^*_{T_2})}_{=c^*_{T_1}}, \underbrace{[c^*_{T_2}, ..., c^*_{T_3})}_{=c^*_{T_2}}, \underbrace{[c^*_{T_V}, ..., c^*_{T_{V+1}})}_{=c^*_{T_V}}$ where the reward function does not change in each segment, thus takes the same optimal choice $c^*_{T_v}$.

We consider an arbitrary segment v and denote the length $\Delta_v = T_{v+1} - T_v$. Furthermore, let define the cumulative gain (or reward) achieved by using TV.EXP3.M strategy within a segment v that the indices are ranging from T_v to $T_{v+1}-1$

$$G_{\text{TV.EXP3.M}}(v) = \sum_{t=T_v}^{T_{v+1}-1}\sum_{c\in S_t} g_t(h_t = c).$$

Taking ln of Eq. (9), we get

$$\ln \frac{W_{t+1}}{W_t} \leq \ln \left(e\alpha + 1 + \frac{\gamma/C}{1-\gamma} \sum_{c \in S_t - S_t(0)} g_t(c) + \frac{(e-2)\left(\frac{\gamma}{C}\right)^2 B}{1-\gamma} \sum_{c=1}^{C} \hat{g}_t(c) \right)$$

$$\leq e\alpha + \frac{\gamma/C}{1-\gamma} \sum_{c \in S_t - S_t(0)} g_t(c) + \frac{(e-2)\left(\frac{\gamma}{C}\right)^2 B}{1-\gamma} \sum_{c=1}^{C} \hat{g}_t(c) \qquad \text{by } 1 + a \leq e^a.$$

Summing over all indices $t = T_v, ..., T_{v+1} - 1$ within a segment v-th:

$$\ln W_{T_{v+1}} - \ln W_{T_v} \leq \Delta_v e\alpha + \frac{\gamma/C}{1-\gamma} \sum_{t=T_v}^{T_{v+1}-1} \sum_{c \in S_t - S_t(0)} g_t(c) + \frac{(e-2)\left(\frac{\gamma}{C}\right)^2 B}{1-\gamma} \sum_{t=T_v}^{T_{v+1}-1} \sum_{c=1}^{C} \hat{g}_t(c)$$

$$\leq \Delta_v e\alpha + \frac{\gamma/C}{1-\gamma} \sum_{t=T_v}^{T_{v+1}-1} \sum_{c \in S_t - S_t(0)} g_t(c) + \frac{(e-2)\left(\frac{\gamma}{C}\right)^2 B}{1-\gamma} \sum_{t=T_v}^{T_{v+1}-1} \sum_{c=1}^{C} \hat{g}_t(c).$$

Let $j = c^*_{T_v} = ... = c^*_{T_{v+1}-1}$ be the optimal choice in the sequence v, we have $\omega_j^{(T_{v+1})} = \omega_j^{(T_{v+1}-1)} \times \exp\left(\frac{B\gamma}{C}\hat{g}_t(j)\right) + \frac{e\alpha}{C} W_{T_{v+1}-1}, \forall j \notin S_{T_{v+1}-1}(0)$ and $\omega_j^{(T_{v+1})} = \omega_j^{(T_{v+1}-1)}, \forall j \in S_{T_{v+1}-1}(0)$

$$\begin{aligned}
\omega_j^{(T_{v+1})} &= \omega_j^{(T_{v+1}-1)} \times \exp\left(\frac{B\gamma}{C}\hat{g}_t(j)\right) + \frac{e\alpha}{C} W_{T_{v+1}-1} \\
&\geq \omega_j^{(T_{v+1}-1)} \times \exp\left(\frac{B\gamma}{C}\hat{g}_t(j)\right) \\
&\geq \omega_j^{(T_{v+1}-2)} \times \exp\left(\frac{B\gamma}{C}\hat{g}_t(j)\right) \exp\left(\frac{B\gamma}{C}\hat{g}_{(T_{v+1}-2)}(j)\right) \\
&\geq \omega_j^{(T_v)} \times \exp\left(\frac{B\gamma}{C} \sum_{t=T_v | t: j \notin S_t(0)}^{T_{v+1}-1} \hat{g}_t(j)\right) \\
&\geq \frac{e\alpha}{C} W_{T_v} \times \exp\left(\frac{B\gamma}{C} \sum_{t=T_v | t: j \notin S_t(0)}^{T_{v+1}-1} \hat{g}_t(j)\right) \\
&\geq \frac{\alpha}{C} W_{T_v} \times \exp\left(\frac{B\gamma}{C} \sum_{t=T_v | t: j \notin S_t(0)}^{T_{v+1}-1} \hat{g}_t(j)\right).
\end{aligned}$$

We now consider the lower bound by taking the optimal (batch) set $A^*_v \subset [C]$ for B elements with the maximum total of reward within the segment v: $\sum_{c \in A^*_v} \sum_{t=1}^{T} g_t(c)$. Since we have the fact that $\sum_{j \in A^*_v} \omega_j^{(T_{v+1})} \geq B \left(\prod_{j \in A^*_v} \omega_j^{(T_{v+1})}\right)^{1/B}$ by Cauchy Swatchz inequality, we continue the above formula

$$\begin{aligned}
\ln\left(W_{T_{v+1}-1}\right) - \ln W_{T_v} &\geq \ln B + \frac{1}{B} \sum_{j \in A^*_v} \ln \omega_j^{(T_{v+1})} - \ln W_{T_v} \\
&= \ln B + \frac{1}{B} \sum_{j \in A^*_v} \ln \frac{\alpha}{C} W_{T_v} + \frac{1}{B} \sum_{j \in A^*_v} \frac{B\gamma}{C} \sum_{t=T_v | t: j \notin S_t(0)}^{T_{v+1}-1} \hat{g}_t(j) - \ln W_{T_v} \\
&= \ln B + \frac{1}{B} \sum_{j \in A^*_v} \ln \frac{\alpha}{C} + \sum_{j \in A^*_v} \frac{\gamma}{C} \sum_{t=T_v | t: j \notin S_t(0)}^{T_{v+1}-1} \hat{g}_t(j) \\
&= \ln \frac{B\alpha}{C} + \sum_{j \in A^*_v} \frac{\gamma}{C} \sum_{t=T_v | t: j \notin S_t(0)}^{T_{v+1}-1} \hat{g}_t(j).
\end{aligned}$$

Combining both the lower bound and upper bound, we get

$$\ln \frac{B\alpha}{C} + \sum_{j\in A_v^*} \frac{\gamma}{C} \sum_{t=T_v|t:j\notin S_t(0)}^{T_{v+1}-1} \hat{g}_t(j) \leq \Delta_v e\alpha + \frac{\gamma/C}{1-\gamma} \sum_{t=T_v}^{T_{v+1}-1} \sum_{c\in S_t - S_t(0)} g_t(c) + \frac{(e-2)\left(\frac{\gamma}{C}\right)^2 B}{1-\gamma} \sum_{t=T_v}^{T_{v+1}-1} \sum_{c=1}^{C} \hat{g}_t(c).$$

Summing over all segments $v = 1....V$

$$\begin{aligned}
V \ln \frac{B\alpha}{C} + \sum_{v=1}^{V} \sum_{j\in A_v^*} \frac{\gamma}{C} \sum_{t=T_v|t:j\notin S_t(0)}^{T_{v+1}-1} \hat{g}_t(j) \leq & Te\alpha + \frac{\gamma/C}{1-\gamma} \sum_{v=1}^{V} \sum_{t=T_v}^{T_{v+1}-1} \sum_{c\in S_t - S_t(0)} g_t(c) \\
& + \frac{(e-2)\left(\frac{\gamma}{C}\right)^2 B}{1-\gamma} \sum_{v=1}^{V} \sum_{t=T_v}^{T_{v+1}-1} \sum_{c=1}^{C} \hat{g}_t(c) \\
\frac{VC}{\gamma} \ln \frac{B\alpha}{C} + \sum_{v=1}^{V} \sum_{j\in A_v^*} \sum_{t=T_v|t:j\notin S_t(0)}^{T_{v+1}-1} \hat{g}_t(j) \leq & \frac{C}{\gamma} Te\alpha + \frac{1}{1-\gamma} \sum_{v=1}^{V} \sum_{t=T_v}^{T_{v+1}-1} \sum_{c\in S_t - S_t(0)} g_t(c) \\
& + \frac{(e-2)\left(\frac{\gamma B}{C}\right)}{1-\gamma} \sum_{v=1}^{V} \sum_{t=T_v}^{T_{v+1}-1} \sum_{c=1}^{C} \hat{g}_t(c) \\
\sum_{v=1}^{V} \sum_{t=T_v|j\in S_t(0)}^{T_{v+1}-1} \sum_{j\in A_v^*} g_t(j) + \sum_{v=1}^{V} \sum_{j\in A_v^*} \sum_{t=T_v|j\notin S_t(0)}^{T_{v+1}-1} \hat{g}_t(j) \leq & \frac{C}{\gamma} Te\alpha + \frac{1}{1-\gamma} \sum_{v=1}^{V} \sum_{t=T_v}^{T_{v+1}-1} \sum_{c\in S_t(0)} g_t(c) \\
& + \frac{1}{1-\gamma} \sum_{v=1}^{V} \sum_{t=T_v}^{T_{v+1}-1} \sum_{c\in S_t - S_t(0)} g_t(c) \\
& + \frac{(e-2)\left(\frac{\gamma B}{C}\right)}{1-\gamma} \sum_{v=1}^{V} \sum_{t=T_v}^{T_{v+1}-1} \sum_{c=1}^{C} \hat{g}_t(c) - \frac{VC}{\gamma} \ln \frac{B\alpha}{C}
\end{aligned} \tag{10}$$

Let us denote the optimal gain over all iterations and all parallel agents $G_{TB}^* = \sum_{v=1}^{V} \sum_{t=T_v}^{T_{v+1}-1} \sum_{j\in A_v^*} g_t(j)$. We also denote the cumulative gain achieved by using TV.EXP3.M algorithms $G_{\text{TV.EXP3.M}} = \sum_{v=1}^{V} \sum_{t=T_v}^{T_{v+1}-1} \sum_{c\in S_t} g_t(c)$. Then, we continue Eq. (10) as

$$G_{TB}^* \leq \frac{C}{\gamma} Te\alpha + \frac{1}{1-\gamma} G_{\text{TV.EXP3.M}} + \frac{(e-2)\left(\frac{\gamma B}{C}\right)}{1-\gamma} \sum_{v=1}^{V} \sum_{t=T_v}^{T_{v+1}-1} \sum_{c=1}^{C} \hat{g}_t(c) - \frac{VC}{\gamma} \ln \frac{B\alpha}{C} \tag{11}$$

where the number of element in a batch $|S_t| = |A_j^*| = B$. Let take expectation both sides of Eq. (11), we have $\mathbb{E}\left[\hat{g}_t(c) \mid S_1, S_2, ..., S_{i-1}\right] = g_t(c)$ from the fact that DepRound [20] selects action c with probability $p_c(t)$, we obtain

$$G_{TB}^* \leq \frac{1}{(1-\gamma)} \mathbb{E}\left[G_{\text{TV.EXP3.M}}\right] + \frac{(e-2)\left(\frac{\gamma B}{C}\right)}{1-\gamma} \sum_{v=1}^{V} \sum_{t=T_v}^{T_{v+1}-1} \sum_{c=1}^{C} g_t(c) + \frac{C}{\gamma} Te\alpha - \frac{VC}{\gamma} \ln \frac{B\alpha}{C}.$$

We finally have

$$(1-\gamma) G_{TB}^* \leq \mathbb{E}\left[G_{\text{TV.EXP3.M}}\right] + (e-2)\left(\frac{\gamma B}{C}\right) \frac{C}{B} G_{TB}^* + \frac{(1-\gamma)}{\gamma} CTe\alpha - \frac{(1-\gamma)VC}{\gamma} \ln \frac{B\alpha}{C} \tag{12}$$

$$\begin{aligned}
G_{TB}^* - \mathbb{E}\left[G_{\text{TV.EXP3.M}}\right] \leq & (e-1)\gamma G_{TB}^* + \frac{(1-\gamma)}{\gamma} CTe\alpha + \frac{(1-\gamma)VC}{\gamma} \ln \frac{C}{B\alpha} \\
\leq & (e-1)\gamma TB + \frac{(1-\gamma)}{\gamma} CTe\alpha + \frac{VC}{\gamma} \ln \frac{C}{B\alpha} - VC \ln \frac{C}{B\alpha} && \text{by } G_{TB}^* \leq TB \\
\leq & (e-1)\gamma TB + \frac{1}{\gamma} CTe\alpha + \frac{VC}{\gamma} \ln \frac{C}{B\alpha}.
\end{aligned} \tag{13}$$

Algorithm 5 TV.EXP3.M algorithm

Input: $\gamma = \sqrt{\frac{C\ln(C/B)}{(e-1)BT}}$, $\alpha = \frac{1}{T}$,C #categorical choice, T #max iteration, B #multiple play

Init $\omega_c = 1, \forall c = 1...C$ and denote $\eta = (\frac{1}{B} - \frac{\gamma}{C})\frac{1}{1-\gamma}$
for $t = 1$ to T **do**
  if $\arg\max_{c\in[C]} \omega_c \geq \eta \sum_{c=1}^{C} \omega(c)$ **then**
    ν s.t. $\frac{\nu}{\eta} = \sum_{\omega_t(c)\geq\nu} \nu + \sum_{\omega_t(c)<\omega_t(c)} \omega_t(c)$
    Set $S_0 = (c : \omega_t(c) \geq \nu)$ and $\omega_t(S_0) = \nu$
  else
    Set $S_0 = \emptyset$
  end if
  Compute $p_t^c = B\left((1-\gamma)\frac{\omega_c}{\sum_{c=1}^{C}\omega_c} + \frac{\gamma}{C}\right), \forall c$
  $S_t = \text{DepRound}\left(B, [p_t^1 p_t^2 .... p_t^C]\right)$
  Observe the reward $g_t(c) = f(h_t = c)$ for $c \in S_t$
  $\hat{g}_t(c) = \frac{g_t(c)}{p_t^c}, \forall c \in S_t$ and $\hat{g}_t(c) = 0$ otherwise
  $\forall c \notin S_0$: update $\omega_c = \omega_c \times \exp\left(B\gamma\hat{g}_t(c)/C\right) + \frac{e\alpha}{C}\sum_{i=1}^{C}\omega_c$,
  $\forall c = S_0$: update $\omega_c = \omega_c + \frac{e\alpha}{C}\sum_{i=1}^{C}\omega_c$
end for

Output: $\mathcal{D}_T$

In Eq. (12), we use the fact that $\sum_{v=1}^{V}\sum_{t=T_v}^{T_{v+1}-1}\sum_{c=1}^{C} g_t(c) \leq \frac{C}{B}G_{TB}^* = \frac{C}{B}\sum_{v=1}^{V}\sum_{t=T_v}^{T_{v+1}-1}\sum_{c\in A^*} g_t(c)$.

Set $\gamma = \min\left\{1, \sqrt{\frac{C\ln(C/B)}{(e-1)BT}}\right\}$ and $\alpha = \frac{1}{T}$. Then, we have the cumulative regret over all iterations and considering the best element in a batch

$$\mathbb{E}\left[R_{TB}\right] = \mathbb{E}\left[\sum_{v=1}^{V}\sum_{t=T_v}^{T_{v+1}-1}\overbrace{\max_{\forall c\leq C} g_t(c)}^{\text{optimal arms}}\right] - \mathbb{E}\left[\sum_{v=1}^{V}\sum_{t=T_v}^{T_{v+1}-1}\overbrace{\max_{\forall c\in S_t} g_t(c)}^{\text{best arm in a batch } S_t}\right] \tag{14}$$

$$\leq \frac{1}{B}G_{TB}^* - \frac{1}{B}\mathbb{E}\left[G_{\text{TV.EXP3.M}}\right] \tag{15}$$

$$\leq \frac{1}{B}\sqrt{\frac{C(e-1)\ln(C/B)}{BT}}T + \frac{1}{B}\frac{\sqrt{C(e-1)BT}}{\sqrt{\ln(C/B)}}e + \frac{1}{B}\frac{VC}{\sqrt{\frac{C\ln(C/B)}{(e-1)BT}}}\ln\frac{CT}{B} \tag{16}$$

$$\leq \left[1 + e + V\right]\sqrt{(e-1)\frac{CT}{B}\ln\frac{CT}{B}} \tag{17}$$

where we obtain Eq. (15) because the best gain should be greater than the average gain of a batch $\sum_{v=1}^{V}\sum_{t=T_v}^{T_{v+1}-1}\max_{\forall c\in S_t} g_t(c) \geq \frac{1}{B}\sum_{v=1}^{V}\sum_{t=T_v}^{T_{v+1}-1}\sum_{c\in S_t} g_t(c)$.

Given the number of changing points in our reward function is bounded $V \ll \sqrt{T}$ and the number of category C is a constant, our regret bound achieves sublinear regret rate with the number of iterations T, i.e., $\lim_{T\to\infty}\frac{\mathbb{E}[R_{TB}]}{T} = 0$.

□

In the above derivation, T refers to the number of bandit updates which is equivalent to the number of t_{ready} in PB2 setting [44].

D.2 Theoretical Derivations for PB2-D

We adapt Lemma 1 in **(author?)** [44] to handle categorical variables. We first restate some notations used in **(author?)** [44]. Let $F_t^{c_t}(x_t)$ be an objective function under a given set of continuous

hyperparameters x_t and categorical variable c_t at timestep t. An example of $F_t^{c_t}(x_t)$ could be the reward for a deep RL agent. When training for a total of T steps, our goal is to maximize the final performance $F_T^{c_T}(x_T)$. We formulate this problem as optimizing the time-varying black-box reward function f_t, over $\mathcal{D}$.

Lemma 4. *Maximizing the final performance F_T of a model with respect to a given continuous hyperparameter schedule $\{x_t\}_{t=1}^T$ and a categorical hyperparameter schedule $\{c_t\}_{t=1}^T$ is equivalent to maximizing the time-varying black-box function $f_t(x_t)$ and minimizing the corresponding cumulative regret $r_t(x_t)$,*

$$\max F_T^{c_T}(x_T) = \max \sum_{t=1}^{T} f_{c_t}(x_t) = \min \sum_{t=1}^{T} r_t(x_t). \tag{18}$$

Proof. At each t_{ready}, we select a categorical variable $c_t \in \{1, ..., C\}$ and a continuous variable $x_t \in \mathcal{D}^{c_t}$. We would emphasize that the continuous variables are conditioned on the choice of c_t. We observe and record noisy observations, $y_t = f_{c_t}(x_t) + \epsilon_t$, where $\epsilon_t \sim \mathcal{N}(0, \sigma^2\mathbf{I})$ for some fixed σ^2. The function f_t represents the change in F_t after training for t_{ready} steps, i.e. $f_{c_t}(x_t) = F_t^{c_t}(x_t) - F_{t-t_{\text{ready}}}^{c_{t-t_{\text{ready}}}}(x_{t-t_{\text{ready}}})$. We define the best choice at each timestep as $x_t^* = \arg\max_{x_t \in \mathcal{D}^{c_t}} f_{c_t}(x_t)$, and $c_t^* = \arg\max_{c_t, x_t} f_{c_t}(x_t)$. The intermediate regret of each decision is defined as $r_t = f_{c_t^*}(x_t^*) - f_{c_t}(x_t)$ where $f_{c_t^*}(x_t^*)$ is an unknown constant.

We have a reward at the starting iteration $F_1^{c_1}(x_1)$ as a constant that allows us to write the objective function as:

$$F_T^{c_T}(x_T) - F_1^{c_1}(x_1) = \underbrace{F_T^{c_T}(x_T) - F_{T-1}^{c_{T-1}}(x_{T-1})}_{f_{c_{T-1}}(x_{T-1})} + ... + \underbrace{F_3^{c_3}(x_3) - F_2^{c_2}(x_2)}_{f_{c_2}(x_2)} + \underbrace{F_2^{c_2}(x_2) - F_1^{c_1}(x_1)}_{f_{c_1}(x_1)}. \tag{19}$$

Therefore, maximizing the left of Eq. (19) is equivalent to minimizing the cummulative regret as follows:

$$\max\left[F_T^{c_T}(x_T) - F_1^{c_1}(x_1)\right] = \max \sum_{t=1}^{T} F_t^{c_t}(x_t) - F_{t-1}^{c_{t-1}}(x_{t-1}) = \max \sum_{t=1}^{T-1} f_{c_t}(x_t) = \min \sum_{t=1}^{T-1} r_t(x_t)$$

where we define $f_{c_t}(x_t) = F_t^{c_t}(x_t) - F_{t-1}^{c_{t-1}}(x_{t-1})$, the regret $r_t = f_{c_t^*}(x_t^*) - f_{c_t}(x_t)$. □

We then restate Theorem 2 in **(author?)** [44] which will be used in deriving the convergence guarantee for PB2-D.

Theorem 5. *(Theorem 2 in* **(author?)** *[44]) Let the domain $\mathcal{D} \subset [0, r]^d$ be compact and convex where d is the dimension and suppose that the kernel is such that $f_t \sim GP(0, k)$ is almost surely continuously differentiable and satisfies Lipschitz assumptions $\forall L_t \geq 0, t \leq \mathcal{T}, \forall j \leq d, p(\sup\left|\frac{\partial f_t(\mathbf{x})}{\partial \mathbf{x}^{(j)}}\right| \geq L_t) \leq ae^{-(L_t/b)^2}$ for some a, b. Pick $\delta \in (0, 1)$, set $\beta_T = 2\log\frac{\pi^2 T^2}{2\delta} + 2d\log rdbT^2\sqrt{\log\frac{da\pi^2T^2}{2\delta}}$ and define $C_1 = 32/\log(1+\sigma_f^2)$, the PB2 algorithm satisfies the following regret bound after T time steps over B parallel agents with probability at least $1-\delta$:*

$$R_{TB} = \sum_{t=1}^{T} f_t(\mathbf{x}_t^*) - \max_{b \leq B} f_t(\mathbf{x}_{t,b}) \leq \sqrt{C_1 T \beta_T \left(\frac{T}{\tilde{N}B} + 1\right)\left(\gamma_{\tilde{N}B} + \left[\tilde{N}B\right]^3 \omega\right)} + 2$$

the bound holds for any block length $\tilde{N} \in \{1, ..., T\}$ and $B \ll T$.

Next, we are going to derive the main theorem of the paper.

Theorem 6. *(Theorem 2 in the main paper) Let the domain for continuous variables $\mathcal{D} \subset [0, r]^d$ be compact and convex where d is the dimension and suppose that the kernel is such that $f_t \sim GP(0, k)$ is almost surely continuously differentiable and satisfies Lipschitz assumptions $\forall L_t \geq 0, t \leq \mathcal{T}, \forall j \leq d, p(\sup\left|\frac{\partial f_t(\mathbf{x})}{\partial \mathbf{x}^{(j)}}\right| \geq L_t) \leq ae^{-(L_t/b)^2}$ for some a, b.*

Assume the reward distributions to change at arbitrary time instants, but the total number of change points is no more than V. Set $\alpha = \frac{1}{T}$, $\gamma = \min\left\{1, \sqrt{\frac{C\ln(C/B)}{(e-1)BT}}\right\}$, $\beta_T = 2\log\frac{\pi^2T^2}{2\delta} + 2d\log rdbT^2\sqrt{\log\frac{da\pi^2T^2}{2\delta}}$, pick $\delta \in (0,1)$ and define $C_1 = 32/\log(1+\sigma_f^2)$, the PB2-D algorithm satisfies the following regret bound after T time steps over B parallel agents with probability at least $1-\delta$:

$$\mathbb{E}\left[R_{TB}\right] \leq [1+e+V]\sqrt{(e-1)\frac{CT}{B}\ln\frac{CT}{B}} + \sqrt{C_1 T_{c_t^*}\beta_{T_{c_t^*}}\left(\frac{T_{c_t^*}}{\tilde{N}B}+1\right)\left(\gamma_{\tilde{N}B} + \left[\tilde{N}B\right]^3\omega\right)} + 2.$$

The bound holds for any $\tilde{N} \in \{1, ..., T_{c_t^*}\}$ and $V \ll \sqrt{T}$.

Proof. We expand the cumulative regret and optimize it using time-varying GP bandit optimization [8, 44]

$$\begin{aligned} R_{TB} &= \sum_{t=1}^{T} f^* - \max_{b=1\ldots B}\sum_{t=1}^{T} f_{c_{t,b}}(x_{t,b}) \\ &= \max_{b=1,\ldots,B}\sum_{t=1}^{T} f_{c^*_{t,b}}(x_{t,b}) - \max_{b=1,\ldots,B}\sum_{t=1}^{T} f_{c_{t,b}}(x_{t,b}) + \sum_{t=1}^{T} f^* - \max_{b=1,\ldots,B}\sum_{t=1}^{T} f_{c^*_{t,b}}(x_{t,b}) \end{aligned}$$

where $b \in \{1, ..., B\}$ is an agent's index being trained in parallel, $f_{c_t^*}(x_t) = \max_{\forall c_t \in \{1,\ldots,C\}} f_{c_t}(x_t)$ and $c_t^* = \arg\max_{\forall c \in \{1,\ldots C\}} f_c(x_t)$ is the optimal categorical choice at iteration t.

We bound the two terms separately as follows. The first term is bounded by Theorem 1. We assume the process of generating the arm c_t's reward $f_{c_t}(.) := f_{c_t}(x_t)$ is by the "adversary" that TV.EXP3.M does not have the direct control on the selection of x_t. Particularly, x_t will be chosen by TV-GP-BUCB (as part of PB2) in Eqn. (3). We take the expectation of the first term to have

$$\begin{aligned} \mathbb{E}\left[\max_{b=1\ldots B}\sum_{t=1}^{T}\underbrace{f_{c^*_{t,b}}(.)}_{\text{pull optimal arm}}\right] - \mathbb{E}\left[\max_{b=1\ldots B}\sum_{t=1}^{T}\underbrace{f_{c_{t,b}}(.)}_{\text{pull arm } c_t}\right] &= \mathbb{E}\left[\sum_{t=1}^{T}\underbrace{f_{c_t^*}(.)}_{\text{pull optimal arm}}\right] - \mathbb{E}\left[\max_{b=1\ldots B}\sum_{t=1}^{T}\underbrace{f_{c_{t,b}}(.)}_{\text{pull arm } c_t}\right] \\ &= \mathbb{E}\left[\tilde{R}_{TB}\right] \\ &\leq [1+e+V]\sqrt{(e-1)\frac{CT}{B}\ln\frac{CT}{B}} \end{aligned}$$

where R_{TB} is cumulative regret of the TV.EXP3.M, defined in Theorem 1.

Assuming the best arm (the best categorical choice) c_t^* can be identified by TV.EXP3.M in Theorem 1. The second term is the regret bound presented in Theorem 2 of the PB2 [44]:

$$\begin{aligned} \sum_{t=1}^{T} f_t^* - \max_{b=1\ldots B}\sum_{t=1}^{T} f_{c^*_{t,b}}(x_{t,b}) &= \sum_{t=1}^{T} f_t^* - \max_{b=1\ldots B}\sum_{t=1}^{T} f_{c_t^*}(x_{t,b}) \\ &\leq \underbrace{\sqrt{C_1 T_{c_t^*}\beta_{T_{c_t^*}}\left(\frac{T_{c_t^*}}{\tilde{N}B}+1\right)\left(\gamma_{\tilde{N}B} + \left[\tilde{N}B\right]^3\omega\right)} + 2}_{\mathcal{O}(PB2)} \end{aligned}$$

where $T_{c_t^*}$ denotes the number of times the optimal category c_t^* (or optimal arm) is selected. We follow **(author?)** [8, 44] to denote a block length $\tilde{N} \in \{1, ...T_{c_t^*}\}$ in which the function does not change significantly. In the above equation, $\omega \in [0,1]$ is a time-varying hyperparameter which is estimated by maximizing the GP log marginal likelihood, B is a batch size or the number of population agents, $\gamma_{\tilde{N}B}$ is the maximum information gain [63] defined within a block $\tilde{N}$ over B parallel agents.

Note that the theoretical result for PB2 comes with the additional smoothness assumption on the kernel k that holds for some (a, b) and $\forall L_t \geq 0$. The kernel satisfies for all dimensions $j = 1, ..., d$

$$\forall L_t \geq 0, t \leq \mathcal{T}, p(\sup \left| \frac{\partial f_t(\mathbf{x})}{\partial \mathbf{x}^{(j)}} \right| \geq L_t) \leq a e^{-(L_t/b)^2}. \tag{20}$$

We combine the two terms to obtain the final regret bound of the PB2-D algorithm:

$$\mathbb{E}\left[R_{TB}\right] \leq [1 + e + V] \sqrt{(e-1)\frac{CT}{B} \ln \frac{CT}{B}} + \sqrt{C_1 T_{c_t^*} \beta_{T_{c_t^*}} \left(\frac{T_{c_t^*}}{\tilde{N}B} + 1 \right) \left(\gamma_{\tilde{N}B} + \left[\tilde{N}B\right]^3 \omega \right)} + 2.$$

The final regret bound is a summation of two sub-linear terms. Therefore, the regret bound grows sublinearly with the number of iterations T, i.e., $\lim_{T\to\infty} \frac{\mathbb{E}[R_{TB}]}{T} = 0$. This is under a common assumption [8, 44] that the time-varying function is correlated, i.e., we have $\omega \to 0$, thus $\tilde{N} \to T$. We also note that when $T \to \infty$, then $T_{c_T^*} \to \infty$ due to Theorem 1.

□

D.3 Gradients

We optimize the GP hyperparameters by maximizing the log marginal likelihood [49]. We fit the GP hyperparameters by maximizing their posterior probability (MAP), $p\left(\sigma_x, \sigma_t \mid \mathbf{X}, \mathbf{t}, \mathbf{y}\right) \propto p\left(\sigma_x, \sigma_t, \mathbf{X}, \mathbf{t}, \mathbf{y}\right)$, which, thanks to the Gaussian likelihood, is available in closed form

$$\mathcal{L} := \ln p\left(\mathbf{y}, \mathbf{X}, \mathbf{t}, \theta\right) = \frac{1}{2}\mathbf{y}^T \left(\boldsymbol{K} + \sigma_y^2 \mathbf{I}_N\right)^{-1} \mathbf{y} - \frac{1}{2} \ln \left|\boldsymbol{K} + \sigma_y^2 \mathbf{I}_N\right| + \ln p_{hyp}\left(\theta\right) + \text{const} \tag{21}$$

where $\mathbf{I}_N$ is the identity matrix in dimension N (the number of points in the training set), and $p_{hyp}(\theta)$ is the prior over hyperparameters. We optimize Eq. (21) with a gradient-based optimizer, providing the analytical gradient to the algorithm.

Our time-varying CoCaBO kernel is defined as

$$k_z(z, z') = (1 - \lambda)\left(k_{xt} + k_{ht}\right) + \lambda k_{xt} k_{ht}$$

where $k_{xt} = k_{\text{Continuous}}(x, x') \times k_{\text{time1}}(t, t')$, $k_{ht} = k_{\text{Categorical}}(h, h') \times k_{\text{time2}}(t, t')$, $k_{\text{Continuous}}(x, x') = \sigma_1 \times \exp\left(-\frac{||x-x'||^2}{l}\right)$, $k_{\text{Categorical}}(h, h') = \frac{\sigma_2}{C} \sum \mathbb{I}(h, h')$, $k_{\text{time1}}(t, t') = (1 - \epsilon_1)^{\frac{|t-t'|}{2}}$ and $k_{\text{time2}}(t, t') = (1 - \epsilon_2)^{\frac{|t-t'|}{2}}$.

Our hyperparameters include $\theta = \{\epsilon_1, \epsilon_2, l, \sigma_1, \sigma_2, \lambda\}$. We need to compute the gradients of $\frac{\partial \mathcal{L}}{\partial \epsilon_1}, \frac{\partial \mathcal{L}}{\partial \epsilon_2}, \frac{\partial \mathcal{L}}{\partial l}, \frac{\partial \mathcal{L}}{\partial \sigma_1}, \frac{\partial \mathcal{L}}{\partial \sigma_2}, \frac{\partial \mathcal{L}}{\partial \lambda}$ as follows:

- The gradient of $\frac{\partial \mathcal{L}}{\partial \epsilon_1}$

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \epsilon_1} &= \frac{\partial \mathcal{L}}{\partial k_z} \times \frac{\partial k_z}{\partial \epsilon_1} \\
\frac{\partial k_z}{\partial \epsilon_1} &= (1 - \lambda)\, k_{\text{Continuous}}(x, x') \frac{\partial k_{\text{time1}}}{\partial \epsilon_1} + \lambda k_{ht} k_{\text{Continuous}}(x, x') \frac{\partial k_{\text{time1}}}{\partial \epsilon_1} \\
\frac{\partial k_{\text{time1}}}{\partial \epsilon_1} &= -\frac{|t - t'|}{2} (1 - \epsilon_1)^{\frac{|t-t'|}{2} - 1}
\end{aligned}$$

- The gradient of $\frac{\partial \mathcal{L}}{\partial \epsilon_2}$

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \epsilon_2} &= \frac{\partial \mathcal{L}}{\partial k_z} \times \frac{\partial k_z}{\partial \epsilon_2} \\
\frac{\partial k_z}{\partial \epsilon_2} &= (1 - \lambda)\, k_{\text{Categorical}}(h, h') \frac{\partial k_{\text{time2}}}{\partial \epsilon_2} + \lambda k_{xt} k_{\text{Categorical}}(h, h') \frac{\partial k_{\text{time2}}}{\partial \epsilon_2} \\
\frac{\partial k_{\text{time2}}}{\partial \epsilon_2} &= -\frac{|t - t'|}{2} (1 - \epsilon_2)^{\frac{|t-t'|}{2} - 1}
\end{aligned}$$

- The gradient of $\frac{\partial \mathcal{L}}{\partial l}$

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial l} =& \frac{\partial \mathcal{L}}{\partial k_z} \times \frac{\partial k_z}{\partial l} \\
\frac{\partial k_z}{\partial l} =& (1-\lambda)\, k_{\text{time1}}(t,t') \frac{\partial k_{\text{Continuous}}}{\partial l} + \lambda k_{ht} k_{\text{time1}}(t,t') \frac{\partial k_{\text{Continuous}}}{\partial l} \\
\frac{\partial k_{\text{Continuous}}}{\partial l} =& \frac{||x-x'||^2}{l^2} k_{\text{Continuous}}(x,x')
\end{aligned}$$

- The gradient of $\frac{\partial \mathcal{L}}{\partial \sigma_1}$

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \sigma_1} =& \frac{\partial \mathcal{L}}{\partial k_z} \times \frac{\partial k_z}{\partial \sigma_1} \\
\frac{\partial k_z}{\partial \sigma_1} =& (1-\lambda)\, k_{\text{time1}}(t,t') \frac{\partial k_{\text{Continuous}}}{\partial \sigma_1} + \lambda k_{ht} k_{\text{time1}}(t,t') \frac{\partial k_{\text{Continuous}}}{\partial \sigma_1} \\
\frac{\partial k_{\text{Continuous}}}{\partial \sigma_1} =& k_{\text{Continuous}}(x,x')
\end{aligned}$$

- The gradient of $\frac{\partial \mathcal{L}}{\partial \sigma_2}$

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \sigma_2} =& \frac{\partial \mathcal{L}}{\partial k_z} \times \frac{\partial k_z}{\partial \sigma_2} \\
\frac{\partial k_z}{\partial \sigma_2} =& (1-\lambda)\, k_{\text{time2}}(t,t') \frac{\partial k_{\text{Categorical}}}{\partial \sigma_2} + \lambda k_{xt} k_{\text{time2}}(t,t') \frac{\partial k_{\text{Categorical}}}{\partial \sigma_2} \\
\frac{\partial k_{\text{Categorical}}}{\partial \sigma_2} =& k_{\text{Categorical}}(h,h')
\end{aligned}$$

- The gradient of $\frac{\partial \mathcal{L}}{\partial \lambda}$

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \lambda} =& \frac{\partial \mathcal{L}}{\partial k_z} \times \frac{\partial k_z}{\partial \lambda} \\
\frac{\partial k_z}{\partial \lambda} =& -(k_{ht} + k_{xt}) + k_{ht} k_{xt}
\end{aligned}$$