Improving planning and MBRL with temporally-extended actions

Palash Chatterjee
Indiana University
palchatt@iu.edu

Roni Khardon
Indiana University
rkhardon@iu.edu

Abstract

Continuous time systems are often modeled using discrete time dynamics but this requires a small simulation step to maintain accuracy. In turn, this requires a large planning horizon which leads to computationally demanding planning problems and reduced performance. Previous work in model-free reinforcement learning has partially addressed this issue using action repeats where a policy is learned to determine a discrete action duration. Instead we propose to control the continuous decision timescale directly by using temporally-extended actions and letting the planner treat the duration of the action as an additional optimization variable along with the standard action variables. This additional structure has multiple advantages. It speeds up simulation time of trajectories and, importantly, it allows for deep horizon search in terms of primitive actions while using a shallow search depth in the planner. In addition, in the model-based reinforcement learning (MBRL) setting, it reduces compounding errors from model learning and improves training time for models. We show that this idea is effective and that the range for action durations can be automatically selected using a multi-armed bandit formulation and integrated into the MBRL framework. An extensive experimental evaluation both in planning and in MBRL, shows that our approach yields faster planning, better solutions, and that it enables solutions to problems that are not solved in the standard formulation.

1 Introduction

Many interesting real life systems evolve continuously with time, but the dynamics of such systems are often modeled using a discrete-time approximation. In these models, time evolves in discrete steps of δ_t called the *timescale* of the system. Simulators used in RL or robotics often use such models to capture physical systems. Setting δ_t to a small value allows the discrete-time models to have a good local approximation. To obtain a trajectory of the system using such models, the dynamics function needs to be evaluated at fixed intervals of δ_t. Because δ_t is small, the number of decisions required to solve even a simple task can be quite large. From the perspective of planning or MBRL, this translates to longer planning horizons (or rollouts) which can limit their effectiveness.

Shooting based planners like Cross Entropy Method (CEM) and Model Predictive Path Integral (MPPI) [Kobilarov, 2012, Botev et al., 2013, Williams et al., 2017, Chua et al., 2018] rely on sampling to estimate the highly rewarding regions of the state space. They are known to perform poorly as the planning horizon increases, especially in environments with noisy dynamics or sparse rewards. This is because in such environments the variance of their estimates increases and they need a large number of samples in order to act reliably [Chatterjee et al., 2023]. In addition, when planning with learned models as in MBRL, longer rollouts can lead to compounding errors [Janner et al., 2019].

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

Simulators like MuJoCo [Todorov et al., 2012] or Arcade Learning Environment [Bellemare et al., 2013] make use of a fixed *frame-skip* in addition to timescale. A frame-skip or *action repeat* of n means that the same action is repeated for n times before the agent is allowed to act again. So the *decision timescale* becomes $n \times \delta_t$. Frame-skip values are normally set heuristically, but as shown by Braylan et al. [2015], the ideal values of frame-skip vary depending on the environment and can heavily influence the performance of the agent. This has led to many efforts in trying to control the decision timescale by learning a policy to control either frame-skip [Durugkar et al., 2016, Lakshminarayanan et al., 2017, Sharma et al., 2017], or timescale [Ni and Jang, 2022]. To the best of our knowledge, none of the previous approaches modeling the decision timescale consider the planning problem or the use of planning in MBRL.

In this paper, we propose to control the decision timescale directly by treating the action duration (δt) as an additional optimization variable. At each *decision step*, the planner optimizes for the standard *primitive action* as well as its duration resulting in a *temporally-extended action*. In contrast to using a fixed frame-skip, the duration for each temporally-extended action can be different allowing the planner to choose actions of varying durations, as can be seen in Figure 4. This provides the planner with an added degree of flexibility and, at the same time, it reduces the search space by constraining the structure of the trajectories that the planner searches over. Further, as a small increase in the temporally-extended planning horizon can translate to a large increase in the primitive planning horizon, this allows the agent to search deeper, enabling it to solve environments which are otherwise too difficult.

Finally, unlike previous work, we do not learn a policy. Our work is in the context of MBRL. We learn a model that predicts the next state and reward due to a temporally-extended action and use the learned model to plan, leading to superior performance as well as faster training. Selecting a suitable range for δt is an important step as it can directly impact the performance of the planner. We show that posing this as a *non-stationary* multi-armed bandit (MAB) problem [Besbes et al., 2014, Lattimore and Szepesvári, 2020] allows us to select the range automatically.

To summarize, our main contributions are as follows:

1. We propose to control the decision timescale directly by letting the planner optimize for primitive action variables as well as the duration of the action, and evaluate this idea both in planning and in MBRL.
2. We show that our approach decreases the search space and the planning horizon for the planner, helping it solve previously unsolvable problems. It also improves the stability of the search by allowing the planner to search deeper and solve problems with sparse rewards.
3. We show that learning temporally-extended dynamics models and selecting a suitable range for δt can be integrated into a single algorithm within the MBRL framework, and that planning using these learned models leads to faster training and superior performance over the standard algorithm.

2 Related Work

Macro-actions [Finkelstein and Markovitch, 1998, Hauskrecht et al., 2013] and *options* [Sutton et al., 1999] are two of the most common methods to introduce temporal abstraction in planning and RL. A macro-action is a usually defined to be sequence of primitive actions that the agent will take. On the other hand, an option consists of a policy, an initiation set and a terminating condition. The initiation set determines when the option can be taken, while its duration depends on when its terminating condition is satisfied. Both macro-actions and options can either be predefined or can be learned from data [Durugkar et al., 2016, Machado et al., 2017a,b, Ramesh et al., 2019].

Frame-skips or action-repeats are simpler forms of macro-actions where each macro-action is essentially a single primitive action repeated multiple times. While frame-skipping has been used as a heuristic in many deep RL solutions [Bellemare et al., 2012, Mnih et al., 2015], Braylan et al. [2015] showed that using a static value of frame-skips across environments can lead to sub-optimal performance. Following this, there have been multiple efforts using the model-free RL framework to make the frame-skip dynamic. Lakshminarayanan et al. [2017] propose to learn a joint policy on an inflated action space of size $n|\mathcal{A}|$ where each primitive action is tied to n corresponding action-repeats. The value of n is usually small to limit the size of the inflated action space. A more

practical approach is to learn a separate network alongside the standard policy to predict the number of action repeats [Sharma et al., 2017, Biedenkapp et al., 2021].

Frame-skips or action-repeats are discrete values and, while they introduce a temporal abstraction for discrete time systems, a continuous representation is both more realistic and more flexible. Some prior work in model-free RL has explored this problem. Ni and Jang [2022] use a modified Soft Actor Critic [Haarnoja et al., 2018] to learn a policy and control the timescale rather than the action duration, with an explicit constraint on the average timescale of the policy. Their formulation of the optimization objective is similar to ours except that they use a linear approximation of the reward function to compute the reward due to a macro-action (details in Section B). Wang and Beltrame [2024] introduce Soft-Elastic Actor Critic to output the duration of the action along with the action itself. However, they modify the reward structure by introducing penalties for the energy and the time taken by each action. Our work is in the planning and MBRL setting. Rather than learn a policy, we use either exact or learned transition and reward functions to plan. Our objective arises naturally from the formulation without the need to add additional constraints or engineer rewards. Further, previous work in model-free RL treat the maximum value of action-repeats or timescale as a hyperparameter. Instead, we use a MAB framework, similar to Li et al. [2018] and Lu et al. [2022], and automatically select the maximum action duration removing the need for tuning an additional hyperparameter.

Finally, in our work, action durations are chosen by the planner, which is different than planning with durative actions [Mausam and Weld, 2008] where the durations are given by the environment.

3 Modeling Temporally-Extended Actions

The standard discrete-time Markov Decision Process (MDP) is specified by $\{\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma\}$ where $\mathcal{S}$ and $\mathcal{A}$ are the state and action spaces respectively and $\gamma \in (0, 1)$ is the discount factor. $s_t \in \mathcal{S}$ and $a_t \in \mathcal{A}$ represents the state and *primitive action* at timestep t. The one-step transition distribution is given by $\mathcal{T}(s_{t+1}|s_t, a_t)$ and the one-step reward distribution is given by $\mathcal{R}(s_t, a_t)$. The expected discounted return is given by $J_t = \mathbb{E}\left[\sum_{i=0}^{D-1} \gamma^i \mathcal{R}(s_{t+i}, a_{t+i})\right]$ where D is the planning horizon. Discrete time simulation of continuous systems assumes that $\mathcal{T}$ and $\mathcal{R}$ capture the transitions and the corresponding reward due to exactly δ_t^{env} duration, where δ_t^{env} is the timescale of the MDP. In cases when $\mathcal{T}$ and $\mathcal{R}$ are unknown to the agent, their empirical estimates, $\hat{\mathcal{T}}$ and $\hat{\mathcal{R}}$, can be learned using data collected by interacting with the MDP. For the following discussion, we overload the term dynamics to mean both the transition and reward function.

Although we eventually care about what *primitive actions* to take in the environment, a planner can work at an abstract level by using *temporally-extended actions*. We use the terms *decision steps* and *execution steps* to distinguish between the number of times the agent outputs an action and the number of *primitive actions* that are actually executed in the environment.

Let us assume that the agent has access to the *primitive dynamics function* (f) which is accurate for all $0 \leq t \leq \delta_t^{\text{env}}$. In the standard setup, the agent uses f at each decision step and outputs an action whose duration is implicitly δ_t^{env}. The return due to a trajectory τ of length $L(\tau)$ is given by $J_1 = \sum_{t=1}^{L(\tau)} \gamma^{t-1} \mathcal{R}(s_t, a_t)$ Here, the number of execution steps is exactly equal to the number of decision steps.

In our proposed framework, the planner explicitly outputs the duration of the action along with the action itself. At decision step k, let the planner output an action a_k and its corresponding duration $\delta t_k \in [\delta t_{\min}, \delta t_{\max}]$. Now the number of execution steps need not necessarily be equal to the number of decision steps. Let $e_k = \lfloor \delta t_k / \delta_t^{\text{env}} \rfloor + \mathbb{1}(\delta t_k \bmod \delta_t^{\text{env}})$ be the number of execution steps associated with decision step k. Further, let $e_{<k} = \sum_{j=1}^{k-1} e_j$ be the total number of execution steps taken prior to decision step k. The return due to a trajectory τ will be

$$J_2 = \sum_{k=1}^{L(\tau)} \gamma^{e_{<k}} \sum_{t=1}^{e_k} \gamma^{t-1} \mathcal{R}(s_{(e_{<k}+t)}, a_{(e_{<k}+t)}) \tag{1}$$

where $e_{<1} = 0$. This formulation provides an approximation that avoids the need for a precise continuous time model of discounting. Note that when $\delta t_k = \delta_t^{\text{env}}$ for all k, then $J_2 = J_1$.

Algorithm 1 One decision step : Action selection using a temporally-extended dynamics function (F) with a generic shooting-based planner

Require: temporally-extended dynamics function (F), current state (s)
initial action distribution (μ_a, var$_a$), number of rollouts (N), planning horizon (D_{TE})

for $i = 1$ to optimization steps **do**
  Sample N action sequences of length D_{TE} using μ_a, var$_a$
  for all N action sequences $\mathbf{a}_{t:t+D_{\text{TE}}} = (a_t, \ldots, a_{t+D_{\text{TE}}})$ **do**
    //a_t includes standard action variables and the action duration
    Simulate trajectory using F from s due to $\mathbf{a}_{t:t+D_{\text{TE}}}$ and compute aggregate reward
  end for
  //update rule is specific to the planner being used.
  Update μ_a, var$_a$ using action sequences and aggregate rewards
end for
return sample action using μ_a, var$_a$

Let $R_k^{\text{TE}} = \sum_{t=1}^{e_k} \gamma_2^{t-1} \mathcal{R}(s_{(e_{<k}+t)}, a_{(e_{<k}+t)})$ be the reward due to the temporally-extended action at decision-step k. Then we have a slightly different view of J_2 where the returns due to a temporally-extended action is discounted based on the number of primitive actions taken prior to the current timestep. Formally,

$$J_3 = \sum_{k=1}^{L(\tau)} \gamma_1^{e_{<k}} R_k^{\text{TE}} = \sum_{k=1}^{L(\tau)} \gamma_1^{e_{<k}} \left(\sum_{t=1}^{e_k} \gamma_2^{t-1} \mathcal{R}(s_{(e_{<k}+t)}, a_{(e_{<k}+t)}) \right) \tag{2}$$

This view allows us to have different discount factors, γ_1 and γ_2, giving us more fine-grained control over the behavior of the agent.

4 Method

We want the planner to have access to a *temporally-extended dynamics function* (F) that can work with temporally-extended actions. If F is available, then using it for planning is straightforward. For example, one can use a shooting-based planner with F to select an action from any state s as shown in Algorithm 1. However, F is usually not readily available. In the standard planning setup, the agent has access to the primitive dynamics function (f) which samples from the one step primitive transition and reward distribution. A simple solution is to wrap f in a loop (as shown in Algorithm A1 in the Appendix) to obtain F_{IP}. This holds as we assumed that f is accurate for all $0 \leq t \leq \delta_t^{\text{env}}$. We call this the *iterative primitive dynamics function*. Although F_{IP} accurately captures F, it fails to facilitate the speed of execution. The time required by F_{IP} to simulate the outcome due to a temporally-extended action is dependent on the duration of the action itself. However, if we had access to F, this would have been a constant time evaluation.

Approximating F : We address this by using neural networks to approximate F using $\hat{F}_{\text{TE}}$. This allows us to predict the next state and reward due to a temporally-extended action from a given state in constant time. For learning this model, we use the framework of MBRL. We collect data by interacting with the environment and use the data to train $\hat{F}_{\text{TE}}$ to predict a distribution over the next states and a point estimate for the reward. The learned model can then be used for planning. For example, Algorithm 1 can be used with $\hat{F}_{\text{TE}}$ instead of F to select actions, as we do in this work.

Selecting $\delta t_{\min}$ and $\delta t_{\max}$: Unlike other action variables, the range of δt does not come from the environment and has to be set by the planner. If the range is too large, the search for the optimal δt becomes harder while a smaller range can limit the search. However, as the range simply influences the spread of the distribution used to sample δt, its value need not be set precisely.

We set $\delta t_{\min} = \delta_t^{\text{env}}$ to be the timescale of the environment. For $\delta t_{\max}$, we consider m exponentially-spaced candidates, where $m = \log_2(T)$ and T is the maximum number of primitive steps defined by the environment. We model each $\delta t_{\max}$ candidate as an arm and pose the selection as a MAB problem. Pulling an arm is equivalent to selecting one of the $\delta t_{\max}$ candidates and using the planner with this value of $\delta t_{\max}$ to collect data for one episode. The aggregate reward observed at the end of an episode indicates the quality of the arm. However, note that the rewards obtained are directly

related to the quality of the learned dynamics model the agent has. As the agent collects more data, the model quality and hence the reward distribution of the arms change, making the problem of arm selection non-stationary. To address this we propose to use an exponential moving average that focuses the reward estimate on recent episodes and hence ameliorates the effect of non-stationarity.

Let $\bar{R}_T$ be the average reward per timestep obtained in episode T and let $\hat{R}_{i,T} = \hat{R}_{i,T-1} + \alpha(\bar{R}_T - \hat{R}_{i,T-1})$ be the exponentially moving average (EMA) of rewards per timestep obtained till episode T due to candidate i. Let $N(i,T)$ be the number of times candidate i was chosen before iteration T. At the start of iteration $T+1$, $\delta t_{\max}$ is selected using the UCB heuristic [Auer et al., 2002] in Equation (3) where we set $\hat{R}_{i,T=0}$ to the mean reward obtained across 5 episodes using the randomly initialized dynamics models, prior to any training.

$$\arg\max_i \left(\hat{R}_{i,T} + c\sqrt{\frac{2\log(T)}{N(i,T)}} \right) \tag{3}$$

Another important question is whether to learn a single dynamics model to be shared by all the arms or learn a separate dynamics model for each arm. Our preliminary experiments, which we describe in Section I.1, suggested that learning a separate dynamics model for each $\delta t_{\max}$ candidate is better.

To summarize, we treat each $\delta t_{\max}$ candidate as a bandit arm. Before each episode, we use a UCB heuristic with EMA of rewards to select an arm and use the selected value of $\delta t_{\max}$ and the corresponding dynamics model to plan for the duration of the episode.

Discussion : Consider two agents - A_{STD} that uses primitive actions and A_{TE} that uses temporally-extended actions and let D_{STD} and D_{TE} be their corresponding planning horizons. Further, for A_{TE}, let $\delta t_{\max} = m \times \delta_t^{\text{env}}$ for some positive integer m.

The planning horizon for the two agents are at different scales and not directly comparable. We introduce the term *maximal primitive horizon* (H) which is the maximal number of primitive actions taken by the agents while planning. For A_{STD}, $H = D_{\text{STD}}$, while for A_{TE}, $H = m \times D_{\text{TE}}$. A fixed value of H ensures that, while planning, A_{TE} does not consider more primitive actions than A_{STD}, at any instant. For practical purposes, A_{TE} will take less primitive actions than A_{STD}. We argue that even in this unfair setting, using temporally-extended actions can be beneficial.

1. Using temporally-extended actions helps to reduce the space of possible trajectories that A_{TE} searches over. For simplicity, let us consider an environment with binary actions. The search space for A_{STD} is 2^H, while for A_{TE}, the search space reduces to $2^{H/m}$. However, this reduction comes at a cost of the flexibility of trajectories as when m is larger the generated trajectories have a more rigid structure.
2. A_{TE} needs to optimize less variables than A_{STD}. In general, if the action space has dimensions $|\mathcal{A}|$, then at each decision step, A_{STD} has to optimize for $H|\mathcal{A}|$ action variables, while A_{TE} will have to optimize for $(H/m)(|\mathcal{A}|+1)$ action variables.
3. $\hat{F}_{\text{TE}}$ evaluates the outcome of temporally-extended actions in constant time, which is similar to that taken by f to evaluate the outcome of a primitive action. As $D_{\text{TE}} < D_{\text{STD}}$ for all practical purposes, using $\hat{F}_{\text{TE}}$ with A_{TE} helps it make a decision faster than A_{STD}.
4. Manipulating the value of m allows us to scale up the maximal planning horizon of A_{TE} without adding any extra variables. This can be useful in environments where rewards are uninformative and a deeper search is required.

5 Experiments

We evaluate the use of temporally-extended actions in planning and in MBRL. We begin by elaborating on our method for both of these settings, before discussing the experiments and results.

Experimental setup : We compare the planning performance of A_{STD} and A_{TE} using CEM, which is a shooting-based planner. CEM maintains a sequence of sampling distributions from which it generates multiple action sequences. For each action sequence, it instantiates multiple particles and computes the trajectory and reward due to each particle. Then, it computes the mean reward per action sequence and uses the top k action sequences to bias its sampling distributions. For evaluation,

Algorithm 2 Using A_{TE} with a fixed $\delta t_{\max}$ as planner for MBRL

Require: dynamics model ensemble($\hat{F}_{\text{TE}}$), batch size (b), number of iterations (n), $\delta t_{\max}$

Initialize empty dataset $\mathcal{D}$
for $i = 1$ to n **do**
  //data collection
  for $t = 1$ to end of episode **do**
    //a_t includes standard action variables and the action duration
    a_t = planner.choose_action(s_t, $\delta t_{\max}$)
    //env.step() implicitly uses temporally-extended dynamics
    s_{t+1}, r_t = env.step(a_t)
    $\mathcal{D}$ = $\mathcal{D}$.append((s_t, a_t, s_{t+1}, r_t))
  end for
  //training
  num_of_batches = len($\mathcal{D}$) / b
  for $j = 1$ to num_of_batches **do**
    for each dynamics model in $\hat{F}_{\text{TE}}$ **do**
      Sample a mini-batch of size b from $\mathcal{D}$
      Compute the loss and gradients using the mini-batch
      Update the model using the gradients
    end for
  end for
end for

we wrap the primitive dynamics function of the simulation environment in an iterative loop using F_{IP} as discussed in the previous section.

For experiments with RL, following Chua et al. [2018], we learn an ensemble of neural networks to approximate the dynamics function. Specifically, A_{STD} learns to approximate f while A_{TE} learns to approximate F. Both the agents use the same network architecture and configuration of parameters. The only difference is that the model learned by A_{TE} has an extra input variable corresponding to δt, in addition to the state and action variables. The data collection and the training procedure for A_{TE} with a fixed $\delta t_{\max}$ in described in Algorithm 2. We use the TS∞ variant from Chua et al. [2018] for planning with the learned ensemble of dynamics models (see Section E for details about planning with ensembles). Note that A_{STD} is equivalent to TS∞ from PETS [Chua et al., 2018].

We consider two variants of our proposed algorithm - $A_{\text{TE}}(\text{F})$ which uses a fixed and manually chosen value of $\delta t_{\max}$ and $A_{\text{TE}}(\text{D})$ which uses the proposed MAB framework to dynamically select the value of $\delta t_{\max}$ at the beginning of each episode. Both variants use $\delta t_{\min} = \delta_t^{\text{env}}$. A key difference between the two is that $A_{\text{TE}}(\text{D})$ maintains a separate dataset and learns a separate temporally-extended dynamics model for each $\delta t_{\max}$ candidate. Specifically, $A_{\text{TE}}(\text{F})$ uses Algorithm 2 exactly, while $A_{\text{TE}}(\text{D})$ has an additional step where it uses the MAB framework to select a $\delta t_{\max}$ value before the start of each episode and then uses the corresponding dataset and model for the duration of the episode. For our discussion, we use A_{TE} to refer to both the variants in general while specifying the variant wherever required.

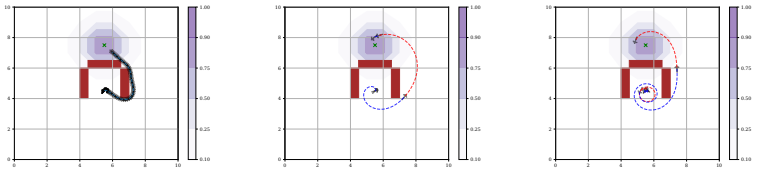
Experimental Evaluation

First, we experiment in a planning regime where the agent has access to the exact dynamics function. In this setting, A_{TE} uses iterative primitive dynamics function (Algorithm A1) for planning. For experiments, we use the Mountain Car environment from Gymnasium [Kwiatkowski et al., 2024], a multi-hill Mountain Car environment from the Probabilistic and Reinforcement Learning Track of the International Planning Competition (IPC) 2023 [Taitler et al., 2024], and the Dubins car environment from Chatterjee et al. [2023]. Then, we experiment in the MBRL setting where $\mathcal{T}$ and $\mathcal{R}$ are not known. In this case, we learn a temporally-extended model as discussed above, by interacting with the environment. We use Cartpole from Gymnasium and Ant, Half Cheetah, Hopper, Reacher, Pusher and Walker from MuJoCo [Todorov et al., 2012]. We run each experiment across 5 different seeds and aggregate the results. Full details of the experimental setup are given in Section E.

The experiments are organized so as to answer a set of questions as outlined below.

Instance	Rewards ↑		Decision steps ↓		Time for an episode ↓		Success Probability ↑	
	Standard	Ours	Standard	Ours	Standard	Ours	Standard	Ours
1	90.98 ± 0.26	91.74 ± 1.55	108.8 ± 0.84	3.0 ± 1.41	4.86 ± 0.19	5.7 ± 1.13	1.00	1.00
2	-0.1 ± 0.01	**88.67 ± 0.65**	300.0 ± 0.0	2.8 ± 1.1	7.98 ± 0.28	5.59 ± 0.91	0.00	**1.00**
3	-0.1 ± 0.01	**86.15 ± 1.11**	300.0 ± 0.0	2.6 ± 0.89	7.94 ± 0.15	5.44 ± 0.76	0.00	**1.00**
4	-0.1 ± 0.01	**66.50 ± 43.54**	300.0 ± 0.0	3.2 ± 1.3	7.73 ± 0.28	6.01 ± 1.05	0.00	**0.80**
5	-0.1 ± 0.01	**83.41 ± 0.54**	300.0 ± 0.0	3.0 ± 0.0	8.0 ± 0.28	5.82 ± 0.2	0.00	**1.00**

Table 1: Results on Multi-hill Mountain Car from IPC 23 across 5 seeds where A_{TE} ($D_{\text{TE}} = 12$) solve all instances while A_{STD} ($D_{\text{STD}} = 175$) can only solve 1/5.



(a) A_{STD}, $|\mathcal{A}|$=2 (b) A_{TE}, $|\mathcal{A}|$=2 (c) A_{TE}, $|\mathcal{A}|$=102

Figure 2: When the number of actions is small (as in a and b), both the agents are able to solve the problem. But when the number of actions increases (c), A_{TE} is still able to solve the problem while A_{STD} fails due to large memory requirements. The shape of the curves is an artifact of the action space in this environment. Note that a constant action in acceleration space yields curved paths. The path in (c) is composed of 4 actions of different durations, as marked by the colors.

Does planning with temporally-extended actions help? Mountain Car has a sparse reward, requiring a large planning horizon to succeed. We compare the performance of A_{STD} and A_{TE} by varying the planning horizon. As shown in Figure 1, $A_{\text{TE}}(D_{\text{TE}} \geq 4)$ solves the environment with a much smaller planning horizon than $A_{\text{STD}}(D_{\text{STD}} \geq 60)$.

Next, we experiment with the multi-hill version of Mountain Car from IPC 2023 (illustrated in Figure A1b in the Appendix). Each instance of the environment increases the difficulty by either adding more hills or altering the surface of the hills. As shown in Table 1, while A_{STD} is able to solve only the first instance, A_{TE} solves all the instances. Note that, even though A_{TE} takes a small number of decision steps, the computation time required to finish an episode is comparable to A_{STD}. This is because A_{TE} uses an iterative version of the dynamics (F_{IP}) for simulation. Another observation is that because the episode terminates when the agent reaches the goal, the number of decision steps is smaller than D_{TE}.

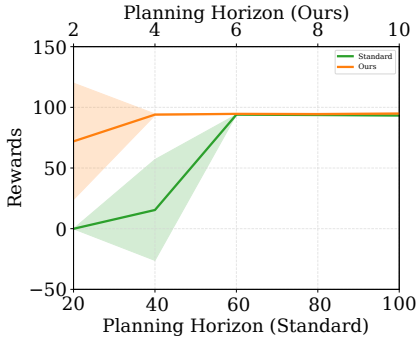


Figure 1: A_{STD} requires a large planning horizon ($D_{\text{STD}} \geq 60$) to succeed in Mountain Car, but A_{TE} using $\delta t_{\max} = 100$ can work with a small planning horizon ($D_{\text{TE}} \geq 4$).

Can temporally-extended actions help transform infeasible problems to feasible ones? We use the Dubins Car environment ($\delta_t^{\text{env}} = 0.2$) and experiment with u-shaped map as shown in Figure 2. This is a challenging configuration, where shooting-based planners often fail, because the car is initially facing the obstacles and a naive forward search hits the obstacles and does not yield useful information. In addition, as the reward is sparse, this map requires the planning horizon to be large. To solve the environment, A_{STD} requires 10,000 samples with a planning horizon of 1000, while A_{TE} requires a planning horizon of 75 and $\delta t_{\max} = 20$ (see detailed discussion in Section G.2).

To increase difficulty, we augment the action space with 100 dummy action variables. Although these variables do not contribute to the dynamics or reward, the agent is unaware of this and has to account for all the action variables. We show that in this difficult setting as well, A_{TE} succeeds while A_{STD}

fails due to large memory requirements. A simple computation shows that A_{STD} needs around 4GB of memory to track the sampled actions, whereas A_{TE} requires just 103MB. In other configurations that reduce the memory requirements (detailed in Section G.2), the search of A_{STD} fails to find the goal. This shows that using temporally-extended actions can often turn infeasible search problems into feasible ones.

How does varying the two discount factors impact the agent's behaviour? The experiments so far use a single value of γ while the proposed formulation in Equation (2) has two discount factors. As we discuss next, while some expectations on the effect of γ_1 and γ_2 are intuitive, a complete characterization is not obvious.

	γ_1	γ_2	Decision Steps	Primitive Steps
Case 1: Fixed γ_1	0.99	1.0	20.2 ± 0.84	121.8 ± 2.95
		0.99	20.2 ± 0.45	122.0 ± 2.00
		0.9	21.6 ± 1.34	121.8 ± 0.84
		0.8	22.2 ± 0.84	122.6 ± 1.52
		0.7	23.2 ± 0.45	122.8 ± 1.64
Case 2: Fixed γ_2	1.0	1.0	19.0 ± 0.71	128.6 ± 1.52
	0.99		20.2 ± 0.84	121.8 ± 2.95
	0.95		18.6 ± 0.55	117.6 ± 0.89
	0.9		16.4 ± 0.55	115.4 ± 0.55

Table 2: When γ_1 is fixed, decreasing γ_2 leads to an increase in number of decision steps. When γ_2 is fixed, decreasing γ_1 leads to a decrease in the number of primitive steps. Results averaged across 5 random seeds.

First, we note that even though D_{TE} is fixed, the primitive planning horizon is dependent on the duration of the actions chosen by the planner. This can result in different levels of discounting in different trajectories. In contrast, in the standard framework, the discounting due to a planning horizon D_{STD} is fixed. This makes predicting the behavior of A_{TE}, upon varying γ_1 and γ_2, difficult. Second, for all decision steps $k > 1$, γ_1 will always be the dominating component in the objective. This is because the exponent term for γ_1 is the number of primitive steps before the decision step k, while the exponent term for γ_2 is proportional to the duration of the temporally-extended action.

To explore the impact of γ_1 and γ_2, we fix one and vary the other. We use the cave-mini map (Figure A1a) in the Dubins Car environment, where the agent has to navigate through multiple obstacles. We have two hypotheses. First, for a fixed γ_1, smaller values of γ_2 will prefer shorter action durations and hence larger number of decision steps. Second, for a fixed γ_2, decreasing γ_1 should prefer longer action durations and a smaller total number of primitive actions. The intuition is that if we reduce γ_1 and the planner does not reduce the number of primitive actions, then the impact of discounting on the overall objective will be higher. Table 2 confirms both these trends.

Do the benefits transfer if we learn the dynamics? Having access to the transition and reward function is not realistic. We would like to be able to learn these as we interact with the environment. For this experiment, we use Ant, Half Cheetah, Hopper, Reacher, Pusher and Walker from MuJoCo along with Cartpole. By default, the MuJoCo simulators use a preset value of frame-skip which vary depending on the environment. This results in the effective timescale being greater than the original timescale. For our experiments, we modify the environments to use a frame-skip of 1.

A_{TE} uses a learned temporally-extended model for planning, while for evaluation we wrap the primitive transition and reward functions in an iterative loop similar to F_{IP}. $A_{\text{TE}}(\text{F})$ uses a fixed $\delta t_{\max}$ throughout the entire learning process while $A_{\text{TE}}(\text{D})$ uses the proposed MAB framework to dynamically select the value of $\delta t_{\max}$ at the start of every episode.

Each experiment is run for a specific number of iterations, where in each iteration, we train the model using multiple mini-batches drawn from the dataset of past transitions, and then use the trained model to act in the environment for one episode. The rewards collected during the episode are used for evaluating the agent's performance. We plot the mean and standard deviation, across 5 different seeds, of the running average of scores obtained as well as the number of decisions made by the agents across training iterations. The main results from our experiments are shown in Figure 3.

First, we review the results across environments before discussing environment specific observations. Overall, A_{TE} performs better than A_{STD} while being faster. $A_{\text{TE}}(\text{F})$, with a suitable selection of $\delta t_{\max}$, results in the best performance. $A_{\text{TE}}(\text{D})$ requires more training iterations than $A_{\text{TE}}(\text{F})$ as it has to spend time exploring all the candidate arms but it is able to catch up to the performance of $A_{\text{TE}}(\text{F})$ which uses a carefully selected value of $\delta t_{\max}$. In addition, A_{TE} is significantly faster than A_{STD} in terms of computation time (see Table A3 in the Appendix). The speed-up for A_{TE} comes from three sources - (i) using temporally-extended actions leads to fewer decision points, (ii) the

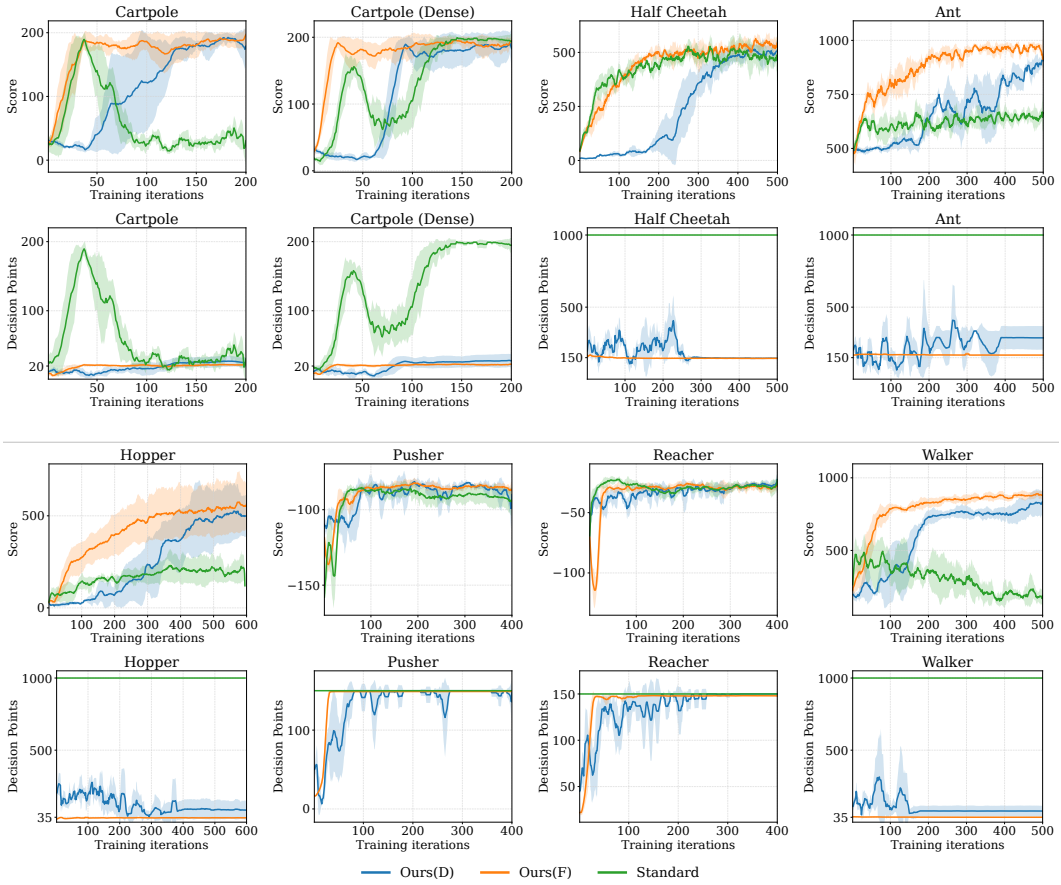


Figure 3: Mean and standard deviation of the running averages (window size=10) of scores and number of decisions taken by the agents in classical control and MuJoCo domains across 5 different seeds. Ours(D) selects $\delta t_{\max}$ automatically using the proposed multi-armed bandit framework while Ours(F) uses fixed $\delta t_{\max}$ for every episode. Using temporally-extended actions results in better performance in many environments while requiring fewer decision points.

reduction in search space, due to temporally-extended actions further reduces planning time and, (iii) a decrease in decision points results in a smaller sized dataset of past transitions that the model needs to learn from, thus reducing the model training time.

In Cartpole, an episode lasts for 200 primitive actions, unless the pole falls which terminates the episode. By default, the agent gets a reward of 1 at each timestep. We find that this reward formulation leads to highly unstable learning using the standard framework. The spike in number of decision points for A_{STD} ($D_{\text{STD}} = 30$) is because the agent learns to keep the pole from falling for the entire duration of the episode, but it soons crashes and the agent is unable to recover. Following Wang et al. [2019], we also experiment with a more informative reward and find that it stabilizes the learning of the standard framework. Both variants of $A_{\text{TE}}(D_{\text{TE}} = 5)$ work well with both the reward functions.

An episode in Ant, Half Cheetah, Hopper and Walker lasts for 1000 primitive actions. By default, the reward per timestep in these environments is not bounded. As our bandit framework uses a UCB heuristic for selecting $\delta t_{\max}$, we modify the reward per timestep to be between 0 and 1. Details of the reward functions are in Section D. In Half Cheetah, $A_{\text{TE}}(D_{\text{TE}} = 15)$ performs similarly to $A_{\text{STD}}(D_{\text{STD}} = 90)$ but is 8x faster. In Ant, Hopper and Walker, A_{TE} significantly outperforms A_{STD}. A_{TE}(F) results in the best performance among agents showing the importance of properly selecting $\delta t_{\max}$. A_{TE}(D) explores all the possible candidate arms before fixating on one, as can be seen from the plots of number of decision points. The value of $\delta t_{\max}$ eventually chosen by A_{TE}(D) is close to A_{TE}(F). Note that A_{TE}(D) does not have the flexibility to choose a value of $\delta t_{\max}$ arbitrarily but has to choose from a fixed list of exponentially spaced $\delta t_{\max}$ candidates.

In Pusher and Reacher, the performance of A_{TE} is again similar to A_{STD}. But in this case, the number of decision points for A_{TE} is also similar to that of A_{STD}, indicating that A_{TE} identifies that a small decision timescale works better here. This illustrates that in cases when a long duration is not suitable, A_{TE}'s performance is still competitive with the standard solution.

How do temporally-extended actions compare to action repeats? Action repeats are a special case of temporally-extended actions where all actions have the same fixed duration. While using temporally-extended actions provides flexibility to an agent in terms of choice of duration for each temporally-extended action, it does add a slight overhead when compared to action repeats. In order to understand whether the agent makes use of the provided flexibility or simply falls back to action repeats, we examine the distribution of action duration chosen by $A_{\text{TE}}(\text{F})$ in Ant, Half Cheetah, Hopper, and Walker *during the course of one episode*. Figure 4 shows the histogram of discretized action durations as action repeats chosen by the agent using the model at the end of training.

While in Ant and Half Cheetah, $A_{\text{TE}}(\text{F})$ mostly chooses actions with similar δt, in Hopper and Walker, the values of δt vary significantly during the episode. This indicates that in environments like Ant and Half Cheetah, the agent can obtain a similar performance by using a fixed δt throughout the episode. However, in environments like Hopper and Walker, the agent benefits from flexibly adapting action duration in each step of execution.

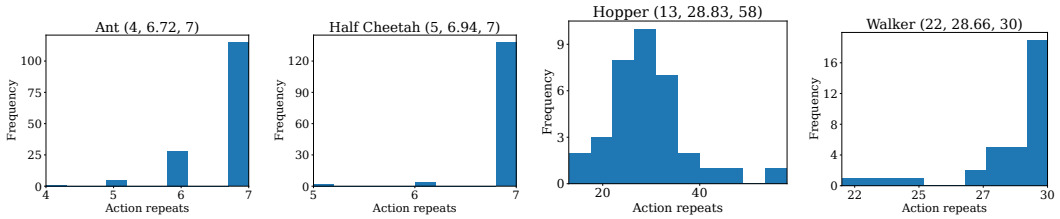


Figure 4: Histogram of action duration in terms of primitive action repeats from the one episode at the end of training for Ant, Half Cheetah, Hopper and Walker. In contrast to using a fixed frame-skip, using temporally-extended actions allows the agent to take actions of varying durations. Numbers in the title indicate the minimum, average and maximum of action repeats taken.

6 Conclusion and Future Work

Using the standard framework of planning with primitive actions provides more granular control but can be computationally expensive. When δ_t^{env} is small, an agent requires a large planning horizon which increases the search space as well as the number of variables it needs to optimize. We propose to use temporally-extended actions where the planner treats the action duration as an additional optimization variable. This decreases the complexity of the search by restricting the search space and reducing the planning horizon, which in turn results in a smaller number of variables for the planner to optimize. Further, we show that, with a suitable $\delta t_{\max}$, learning temporally-extended dynamics models using MBRL can be faster while outperforming the standard framework in many environments. Finally, rather than setting $\delta t_{\max}$ as a hyperparameter, a MAB framework can be used to dynamically select it. This is slower to converge but provides the same performance benefits as a carefully chosen value of $\delta t_{\max}$. Our proposed approach does have several potential limitations. First, using temporally-extended actions does not guarantee improvement in every situation since the generated trajectories are not as flexible as those due to primitive actions. Second, the proposed MAB formulation requires fixed D_{TE} for every bandit arm. Dynamic selection of both $\delta t_{\max}$ and D_{TE} together is left as future work.

Acknowledgements

We would like to thank the reviewers for their time and valueable feedback. This work was partly supported by NSF under grant 2246261. Some of the experiments in this paper were run on the Big Red computing system at Indiana University, supported in part by Lilly Endowment, Inc., through its support for the Indiana University Pervasive Technology Institute.

References

Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2):235–256, 2002.

M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, jun 2013.

Marc Bellemare, Joel Veness, and Michael Bowling. Investigating contingency awareness using atari 2600 games. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 26, pages 864–871, 2012.

Omar Besbes, Yonatan Gur, and Assaf Zeevi. Stochastic multi-armed-bandit problem with non-stationary rewards. *Advances in neural information processing systems*, 27, 2014.

André Biedenkapp, Raghu Rajan, Frank Hutter, and Marius Lindauer. Temporl: Learning when to act. In *International Conference on Machine Learning*, pages 914–924. PMLR, 2021.

Zdravko I Botev, Dirk P Kroese, Reuven Y Rubinstein, and Pierre L'Ecuyer. The cross-entropy method for optimization. In *Handbook of statistics*, volume 31, pages 35–59. Elsevier, 2013.

Alex Braylan, Mark Hollenbeck, Elliot Meyerson, and Risto Miikkulainen. Frame skip is a powerful parameter for learning to play atari. In *Workshops at the twenty-ninth AAAI conference on artificial intelligence*, 2015.

Palash Chatterjee, Ashutosh Chapagain, Weizhe Chen, and Roni Khardon. Disprod: differentiable symbolic propagation of distributions for planning. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, pages 5324–5332, 2023.

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

Ishan P Durugkar, Clemens Rosenbaum, Stefan Dernbach, and Sridhar Mahadevan. Deep reinforcement learning with macro-actions. *arXiv preprint arXiv:1606.04615*, 2016.

Lev Finkelstein and Shaul Markovitch. A selective macro-learning algorithm and its application to the nxn sliding-tile puzzle. *Journal of Artificial Intelligence Research*, 8:223–263, 1998.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

Milos Hauskrecht, Nicolas Meuleau, Leslie Pack Kaelbling, Thomas L Dean, and Craig Boutilier. Hierarchical solution of markov decision processes using macro-actions. *arXiv preprint arXiv:1301.7381*, 2013.

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

Marin Kobilarov. Cross-entropy motion planning. *The International Journal of Robotics Research*, 31(7):855–871, 2012.

Ariel Kwiatkowski, Mark Towers, Jordan Terry, John U. Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Hannah Tan, and Omar G. Younis. Gymnasium: A standard interface for reinforcement learning environments. 2024. URL https://arxiv.org/abs/2407.17032.

Aravind Lakshminarayanan, Sahil Sharma, and Balaraman Ravindran. Dynamic action repetition for deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

Tor Lattimore and Csaba Szepesvári. *Bandit algorithms*. Cambridge University Press, 2020.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18(185):1–52, 2018.

Shiyin Lu, Yu-Hang Zhou, Jing-Cheng Shi, Wenya Zhu, Qingtao Yu, Qing-Guo Chen, Qing Da, and Lijun Zhang. Non-stationary continuum-armed bandits for online hyperparameter optimization. In *Proceedings of the fifteenth ACM international conference on web search and data mining*, pages 618–627, 2022.

Marlos C Machado, Marc G Bellemare, and Michael Bowling. A laplacian framework for option discovery in reinforcement learning. In *International Conference on Machine Learning*, pages 2295–2304. PMLR, 2017a.

Marlos C Machado, Clemens Rosenbaum, Xiaoxiao Guo, Miao Liu, Gerald Tesauro, and Murray Campbell. Eigenoption discovery through the deep successor representation. *arXiv preprint arXiv:1710.11089*, 2017b.

Mausam and Daniel S. Weld. Planning with durative actions in stochastic domains. *Journal of Artificial Intelligence Research*, 31:33–82, 2008.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Tianwei Ni and Eric Jang. Continuous control on time. In *ICLR 2022 Workshop on Generalizable Policy Learning in Physical World*, 2022.

Rahul Ramesh, Manan Tomar, and Balaraman Ravindran. Successor options: An option discovery framework for reinforcement learning. *arXiv preprint arXiv:1905.05731*, 2019.

Sahil Sharma, Aravind Srinivas, and Balaraman Ravindran. Learning to repeat: Fine grained action repetition for deep reinforcement learning. *arXiv preprint arXiv:1702.06054*, 2017.

Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

Ayal Taitler, Ron Alford, Joan Espasa, Gregor Behnke, Daniel Fišer, Michael Gimelfarb, Florian Pommerening, Scott Sanner, Enrico Scala, Dominik Schreiber, et al. The 2023 international planning competition, 2024.

Cem Tekin and Mingyan Liu. Online learning of rested and restless bandits. *IEEE Transactions on Information Theory*, 58(8):5588–5611, 2012.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Siqi Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy Lillicrap, Nicolas Heess, and Yuval Tassa. dm_control: Software and tasks for continuous control. *Software Impacts*, 6:100022, 2020. ISSN 2665-9638. doi: https://doi.org/10.1016/j.simpa.2020.100022.

Dong Wang and Giovanni Beltrame. Deployable reinforcement learning with variable control rate. *arXiv preprint arXiv:2401.09286*, 2024.

Tingwu Wang, Xuchan Bao, Ignasi Clavera, Jerrick Hoang, Yeming Wen, Eric Langlois, Shunshi Zhang, Guodong Zhang, Pieter Abbeel, and Jimmy Ba. Benchmarking model-based reinforcement learning. *arXiv preprint arXiv:1907.02057*, 2019.

Peter Whittle. Restless bandits: Activity allocation in a changing world. *Journal of applied probability*, 25(A):287–298, 1988.

Grady Williams, Andrew Aldrich, and Evangelos A Theodorou. Model predictive path integral control: From theory to parallel computation. *Journal of Guidance, Control, and Dynamics*, 40(2): 344–357, 2017.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: *In the abstract, we claim that using temporally-extended actions adds additional structure to the search and can thus improve planning and learning using MBRL. In the paper, we showcase this empirically by performing extensive experiments.*

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: *In the conclusion, we acknowledge that our method has several potential limitations. The proposed framework does not guarantee improvement in every domain as also noted in MBRL experiments with Reacher and Pusher. Secondly, the meta algorithm for selecting $\delta t_{\max}$ assumes that planning horizon is fixed. A more robust solution will select more $\delta t_{\max}$ and planning horizon simultaneously.*

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [NA]

   Justification: *There are no theoretical results in the paper.*

   Guidelines:

   - The answer NA means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: *The environment specifications along with the necessary hyperparameters are described in the Appendix.*

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: *The code can be found at https://github.com/pecey/MBRL-with-TEA/*

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (https://nips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://nips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: *We provide the relevant experimental configuration in Section E*

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: *We report the mean and the standard deviation across 5 seeds for each experiment.*

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: *We provide the details regarding the computational resources in Section F*

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

   Answer: [Yes]

   Justification: *Our research conforms to the NeurIPS Code of Ethics.*

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [NA]

    Justification: *The paper presents foundational research that is not tied to particular application. Broader impact concerns are hence generic.*

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.

- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: *We are not releasing any data or models as a part of this research.*

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: *All the code required for this research was written by using standard open source frameworks.*

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: *We are not releasing any new assets.*

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: *This work does not involve crowdsourcing or research with human subjects.*

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: *This work does not involve crowdsourcing or research with human subjects.*

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.

- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [NA]

    Justification: *The paper does not use LLMs in any form.*

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (https://neurips.cc/Conferences/2025/LLM) for what should or should not be described.

Appendix

Table of Contents

A List of Symbols

δ_t	Base timescale of the system.
δ_t^{env}	Decision timescale of the environment = Frameskip $\times$ δ_t.
δt	Action duration chosen by the planner.
$\delta t_{\min}, \delta t_{\max}$	Range for the action duration.
f	One-step dynamics function. $\mathcal{S} \times \mathcal{A} \to \mathcal{S}$
F_{IP}	Iterative primitive transition function.
F	Temporally extended dynamics function. $\mathcal{S} \times (\mathcal{A} \cup \delta t) \to \mathcal{S}$
$\hat{F}_{\text{IP}}$	Indirectly approximates F. Learns a one-step model to approximate f and then iterates over it.
$\hat{F}_{\text{TE}}$	Directly approximates F. Learns a model to predict the next state due to a temporally-extended action.
A_{STD}	TS∞ variant of the CEM agent from PETS [Chua et al., 2018].
$A_{\text{TE}}(\text{F})$	Proposed agent that used a fixed and manually chosen value of $\delta t_{\max}$.
$A_{\text{TE}}(\text{D})$	Proposed agent that used a MAB framework to dynamically select $\delta t_{\max}$ before every episode.
D_{STD}	Planning horizon used by the standard agent.
D_{TE}	Planning horizon used by the proposed agent.

B Connection to Ni and Jang [2022]

Ni and Jang [2022] propose to learn a policy that outputs the action variables as well as the time scale. They have a similar looking objective. In this section, we compare our objective to theirs. First, we repeat Equation (1) which computes the returns of a trajectory τ using the proposed framework:

$$J(\tau) = \sum_{k=1}^{L(\tau)} \gamma^{e_{<k}} \sum_{t=1}^{e_k} \gamma^{t-1} \mathcal{R}_{(e_{<k}+t)}.$$

A similar objective is used by Ni and Jang [2022]. However, it is important to note that they control the timescale while we keep the timescale fixed and control the action duration. Let $\delta_k \in [\delta_{\min}, \delta_{\max}]$ be the timescale associated with the action at decision step k. For simplicity, they assume that the timescale is in integers (in physical unit of seconds), which means that δ_k is also the number of execution steps associated with the action. Using this, their objective reduces to

$$J(\tau) = \sum_{k=1}^{L(\tau)} \gamma^{e_{<k}} \mathcal{R}(s_{e_{<k}+1}, a_k)\delta_k \tag{4}$$

where $\mathcal{R}(s_{e_{<k}+1}, a_k)\delta_k$ is a linear approximation of the reward function. So, the objective used by Ni and Jang [2022] can be viewed as an approximation of Equation (1) where the reward function due to a temporally-extended action has been replaced by a linear approximation.

Algorithm A1 F_{IP} : Iterative primitive dynamics function

Require: primitive dynamics function (f), current state (s_k),
temporally-extended action (a_k), duration of action (δt_k)

repeats $= \lfloor \delta t_k / \delta_t^{\text{env}} \rfloor$
for $i = 1$ to repeats **do**
  $s_k = f(s_k, a_k, \delta_t^{\text{env}})$ ▷ Control n
end for
$s_{k+1} = f(s_k, a_k, \delta t_k \bmod \delta_t^{\text{env}})$ ▷ Control δ_t
return s_{k+1}

C Iterative Primitive Dynamics Function

In most cases we have access to the primitive dynamics function (f). A trivial way of making f work with temporally-extended actions is to wrap it in a loop. For the computation to be exact, f should be explicitly dependent on time. If it is implicit, performing the computation at line 5 of Algorithm A1 will not be possible and the algorithm computes the number of action-repeats instead.

The reward is simply aggregated in the loop and is not shown explicitly here. Note that for a given environment, f is fixed, while s_k, a_k and δt_k are dependent on the timestep.

D Environment Overview

In this section, we provide an overview of the environments used in our experiments. Table A1 lists the dimensionality of the observation and the action space of the environments, the reward functions along with the episode length (which is equivalent to the maximum number of primitive steps allowed).

Domain	nS	nA	Episode steps	Reward Function
Dubins Car	5	2	300	$100 \times \mathbb{1}_{\text{distance from goal} < 0.5} - 10 \times \mathbb{1}_{\text{collision = True}}$
Cartpole	4	1	200	$\mathbb{1}_{x,\theta \text{ within bounds}}$
Cartpole (Dense)	4	1	200	$\cos(\theta_t) - 0.001x_t^2$
Mountain Car	2	1	200	$\mathbb{1}_{x_t=\text{goal}} \times 100 - 0.1a_t^2$
IPC Mountain Car	2	1	500	$\mathbb{1}_{x_t=\text{goal}} \times 100 - 0.1a_t^2$
Reacher	17	7	150	$-\text{dist(finger, obj)} - a_t^2$
Pusher	23	7	150	$-0.5 \times \text{dist(finger, obj)} - \text{dist(object, goal)} - 0.1a_t^2$
Ant	27	8	1000	$\text{sigmoid}(\dot{x_t})$ if $\dot{x_t} \leq 0.5$ else 1
Half Cheetah	18	6	1000	$\text{sigmoid}(\dot{x_t})$ if $\dot{x_t} \leq 10$ else 1
Hopper	11	3	1000	$(\text{sigmoid}(\dot{x_t})$ if $\dot{x_t} \leq 2$ else $1) \times$ healthy reward
Walker	17	6	1000	$1/2((\text{sigmoid}(\dot{x_t})$ if $\dot{x_t} \leq 1$ else $1) +$ healthy reward)

Table A1: State and action space, maximum episode lengths and reward functions for different environments.

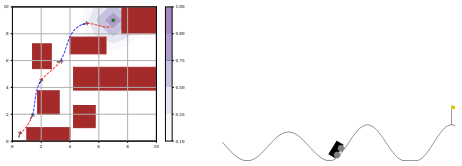
Details for Dubins Car environment : We use the Dubins Car environment of Chatterjee et al. [2023]. The state space comprises of the x and y co-ordinates, and the orientation (θ) of the car, along with the current linear (v) and angular velocity (ω) of the car and the action space is the change in linear velocity (Δv) and angular velocity $(\Delta\omega)$. An episode terminates if the agent reaches the goal or if it takes 300 primitive steps.

D.1 MuJoCo environments

Controlling δ_t^{env} : Each environment defined in MuJoCo has its own value of δ_t which is defined in the corresponding XML file along with the integrator to be used. Further, each environment has a predefined frame-skip which is defined in the constructor of the corresponding class. The combination of these two control δ_t^{env} in MuJoCo. In order to operate at the base δ_t, we simply set the frame-skip value to 1.

Reward functions : The default reward functions in MuJoCo are not bounded. For the MAB framework, we require a bounded reward function. The problem is amplifed in Ant, Half Cheetah, Hopper and Walker, where the rewards over an episode can be very large. Therefore, for these environments, we adapt the bounded reward functions from DeepMind Control Suite [Tunyasuvunakool et al., 2020], so that the agents always get a reward between 0 and 1, with the specifics depending on the environment.

Maximum episode lengths : MuJoCo environments have a predefined value of frame-skip. The default episode length is with respect to this frame-skip. For example, Half Cheetah has a frame-skip of 5 and a episode length of 1000. An equivalent episode in Half Cheetah with a frame-skip of 1 should be of length 5000. However, we limit the episode to 1000 as otherwise A_{STD} would have taken too long to train. Even with episode lengths of 1000, A_{STD} requires more than 40 hours to train (Table A3).



(a) (b)

Figure A1: (a) An example of A_{TE} solving the cave-mini map ($\gamma_1 = 0.99, \gamma_2 = 1.0, \delta t_{\max} = 20$). (b) An instance of the IPC Multi-hill Mountain Car.

E Experimental Details

Among the set of hyperparameters in our framework, the two crucial ones are the range of action duration, which is specified by $\delta t_{\min}$ and $\delta t_{\max}$, and the planning horizon. Rather than controlling both these values, $\delta t_{\min}$ is always set to δ_t^{env}, and modifying $\delta t_{\max}$ controls the range.

Details for Cartpole, Mountain Car and MuJoCo : For the environments Cartpole and Mountain Car, we use values of planning horizon from prior work for the standard framework, and adjust the depth for A_{TE} by exploring related values. We could not do this for MuJoCo-based environments since we set the frameskip parameter to 1. Rather we searched over some potential values for the planning horizon for the standard framework, and $\delta t_{\max}$ and planning horizon for our framework and chose the configuration with the best performance. The other hyperparameters related to online training have been borrowed from Chua et al. [2018].

Domain	Range of action duration		Planning horizon		Learning Rate
	Standard	Ours	Standard	Ours	
Dubins Car [u-shaped map]	0.2	[0.2-20]	1000	75	-
Dubins Car [cave-mini map]	0.2	[0.2-2]	120	50	-
Cartpole	1	[1-10]	30	3	1e-3
Mountain Car	1	[1-100]	100	10	-
IPC Mountain Car	1	[1-125]	175	12	-
Reacher	0.01	[0.01, 0.2]	25	5	1e-3
Pusher	0.01	[0.01, 0.2]	25	5	1e-3
Half Cheetah	0.01	[0.01, 0.07]	90	15	1e-3
Ant	0.01	[0.01, 0.07]	70	15	3e-4
Hopper	0.002	[0.002, 0.125]	50	15	3e-4
Walker	0.002	[0.002, 0.0625]	70	15	3e-4

Table A2: Hyper-parameters for different environments. Range of action duration is for the fixed variant of the algorithm.

Details for Dubins Car environment : The default timescale for the environment is 0.2 while $\delta t_{\max}$ for our formulation is 20. Setting $\delta t_{\max}$ to such a large value allows us to have the same value across maps and just allow more time for optimization. The planning horizon needs to be tuned for different maps. Table A2 contains the planning horizon for the maps used in the experiments. Note that for our experiment with the cave-mini map, we reduce $\delta t_{\max}$ to prevent the agent from completing the map in a few decision steps.

Model learning : For MBRL, we use the same model architecture as Chua et al. [2018]. We learn an ensemble of 5 models, where each model is a fully connected neural network. For Ant, Half Cheetah and Hopper, the model has 4 hidden layers while for all other environments, it has 3 hidden layers. Each hidden layer has 200 neurons. The learning rates for each environment are given in Table A2.

Planning with an ensemble of models : We use the PETS framework from Chua et al. [2018]. They introduce Trajectory Sampling (TS) which uses CEM as the planner. They maintain a sequence of sampling distributions from which N action sequences are generated, and for every action sequence, they create P particles. Each particle is propagated using a particular member of the ensemble. For propagating particle p, TS1 uniformly resamples an ensemble member at each time step, while TS∞ samples an ensemble member before an episode and uses that to propagate particle p for the entire duration of the episode. The mean reward is computed for each action sequence. Then, the mean and

Algorithm A2 Planning using TS∞ with an ensemble of dynamics model.

Require: current state (s), dynamics model ensemble$(\hat{f} = \{\hat{f}_1, \ldots, \hat{f}_K\})$, planning horizon (D), initial action distribution (μ_a, var_a)

for $i = 1$ to number of updates **do**
  Sample N action sequences $\{a_1^N, a_2^N, \ldots, a_D^N\}$ using μ_a, σ_a.
  for $n = 1$ to N **do**
    Initialize P particles = $\{a_1^{n,P}, a_2^{n,P}, \ldots, a_D^{n,P}\}$.
    for $p = 1$ to P **do**
      `//propagate each particle using one model from ensemble`
      $j = p$ `mod` K
      Propagate particle p using $s_{t+1}^{n,p} = \hat{f}_j(s_t^{n,p}, a_t^{n,p})$ and compute aggregate reward.
    end for
    Compute mean reward for action sequence using $\sum_{t=1}^{D} \sum_{p=1}^{P} r(s_t^{n,p}, a_t^{n,p})/P$
  end for
  Choose the top k action sequences and recompute μ_a, var_a
end for
Execute first action from the optimal action sequence.

the variance of the top k action sequences gives the updated sampling distribution. Formally,

$$\mu_a = \frac{1}{k} \sum_{a \in \mathcal{K}} a \tag{5}$$

$$\text{var}_a = \frac{1}{k} \sum_{a \in \mathcal{K}} (a - \mu_a)^2 \tag{6}$$

where $\mathcal{K}$ is the set of top k action sequences. This process is repeated for a fixed number of steps. For our experiments, we learn an ensemble of 5 feed-forward networks and use the TS∞ variant with $P = 20$ for planning. In Algorithm A2, we present the pseudocode for using TS∞ with an ensemble of dynamics models.

F Computational Resources

Each MBRL experiment was performed on a single node with a single GPU (using a mix of V100 and A100), single CPU (AMD EPYC 7742) and 64GB of RAM. The time required for training for different algorithms are in Table A3. We note that although the experiments were done on two different types of GPUs, the running times for both of them are comparable.

Env	Standard	Ours (F)	Ours (D)
Cartpole	0.65	**0.4**	**0.5**
Half Cheetah	45	**5**	**5.5**
Ant	41	**4**	**7**
Hopper	40	**4**	**6.5**
Reacher	2.6	**1.6**	**1.5**
Pusher	2.6	**2**	**1.5**
Walker	36.8	**3.1**	**5.8**

Table A3: Approximate time (in hours) required to finish training for each environment.

In Reacher and Pusher, the rollout length is mostly same for the standard algorithm and our proposed algorithm, but the planning horizon for the standard algorithm is 25 while for our proposed algorithm is 15, resulting in shorter training times.

In Ant, Half Cheetah, Hopper and Walker A_{TE} is significantly faster than A_{STD}. The dynamic variant of our proposed algorithm takes slightly longer than the fixed variant as the rollout length of the dynamic variant is longer on average than the rollout length of the fixed variant.

G Additional Planning Experiments

G.1 Experiments with IPC Mountain Car

How does the performance change if the increase the planning horizon? We vary the planning horizon of the agents and compare the performances across instances in Table A4. For the easier problems, a small planning horizon is sufficient, but for the difficult instances, a deeper search is required. The performance of the planner remains relatively stable as the planning horizon increases.

	D	Instance 1	Instance 2	Instance 3	Instance 4	Instance 5
A_{STD}	60	37.49 ± 51.36	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0
	80	74.54 ± 41.7	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0
	100	91.93 ± 0.41	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0
	125	90.92 ± 0.22	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0
	150	91.07 ± 0.18	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0
	175	91.12 ± 0.3	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0
	200	91.11 ± 0.27	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0
	225	91.2 ± 0.32	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0
	250	91.01 ± 0.12	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0
	275	90.98 ± 0.11	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0
	300	90.91 ± 0.1	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0
A_{TE}	2	-0.01 ± 0.01	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0
	5	73.52 ± 43.45	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0
	8	92.58 ± 1.01	90.99 ± 0.7	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0
	10	71.43 ± 46.05	89.79 ± 1.28	87.27 ± 1.86	68.11 ± 42.49	85.02 ± 0.26
	12	92.41 ± 1.82	90.11 ± 1.19	87.44 ± 1.52	86.91 ± 1.53	85.45 ± 0.25
	15	92.33 ± 2.0	90.52 ± 0.6	87.91 ± 0.8	87.51 ± 0.98	85.19 ± 1.15
	20	92.52 ± 1.46	90.12 ± 1.36	87.74 ± 1.99	87.08 ± 2.57	86.43 ± 0.31
	25	92.39 ± 1.85	90.54 ± 1.06	88.39 ± 1.36	87.71 ± 1.61	87.06 ± 0.46
	30	72.07 ± 45.41	91.08 ± 0.37	88.94 ± 0.49	68.0 ± 45.24	86.08 ± 1.81
	35	51.48 ± 57.79	90.91 ± 1.15	89.28 ± 0.2	89.06 ± 0.81	67.07 ± 45.5
	40	71.21 ± 48.64	90.12 ± 1.42	89.75 ± 2.19	87.38 ± 2.38	67.86 ± 45.2

Table A4: Performance of A_{STD} and A_{TE} in various instances of IPC Multi-hill Mountain Car as the planning horizon is increased.

G.2 Experiments with U-shaped Maps in Dubins Car

We experiment with varying depths using u-shaped maps in the Dubins Car environment. The problem is not trivial as the car faces the obstacle and it needs to first turn around and then find a path the goal, which gives the agent a reward of 100. Every collision with an obstacle gets a penalty of 10. An episode terminates when the agent reaches the goal or when the agent takes 300 primitive steps.

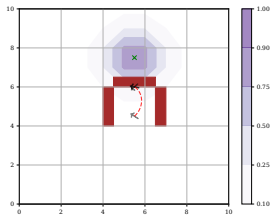
To be successful in this setting, we observe that A_{STD} requires a planning horizon of 1000 and A_{TE} requires a planning horizon of 75. Both the agents use 10,000 samples and 50 optimization steps for each decision. A_{TE} succeeds but not in all runs. We look at the failure cases for A_{TE} and have an interesting observation. For planning horizon of 75 and 100, the failure is due to the fact that the agent runs out of primitive actions, while for planning horizon 50, the failing scenario corresponds to the agent not being able to find a path out of the obstacles region. This means that if we allow the episodes to run for longer, the former failure can be mitigated but the latter cannot. The detailed results are in Table A5 and the failure cases for A_{TE} are shown in Figure A2.

	D	Rewards ↑	Decision Steps ↓	Decision time ↓	Time for an episode ↓	P(success) ↑
A_{STD}	250	-9.95 ± 0.0	300.0 ± 0.0	0.13 ± 0.0	413.46 ± 4.16	0
	500	-9.95 ± 0.0	300.0 ± 0.0	0.29 ± 0.0	806.97 ± 8.82	0
	750	32.04 ± 62.17	213.8 ± 118.03	0.42 ± 0.02	851.53 ± 466.61	0.4
	1000	100.0 ± 0.0	90.8 ± 20.17	0.56 ± 0.02	480.38 ± 102.06	1
A_{TE}	25	-497.53 ± 137.16	4.0 ± 0.0	2.31 ± 0.09	14.1 ± 0.43	0
	50	16.32 ± 187.12	4.2 ± 0.45	3.66 ± 0.1	20.03 ± 1.59	0.8
	75	80.0 ± 44.72	4.4 ± 1.14	5.08 ± 0.24	27.39 ± 5.84	0.8
	100	80.0 ± 44.72	4.4 ± 1.14	6.57 ± 0.21	34.56 ± 7.79	0.8

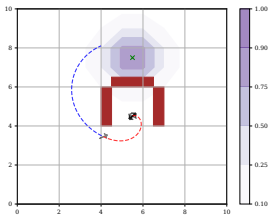
Table A5: Performance of A_{STD} and A_{TE} on the u-shaped map in Dubins Car as the planning horizon is varied. Both agents use 10,000 samples and 50 optimization steps.

	D	#samples	Rewards ↑	Decision Steps ↓	Decision time ↓	Time for an episode ↓	P(success) ↑
A_{STD}	250	10000	-9.95 ± 0.0	300.0 ± 0.0	0.66 ± 0.0	580.85 ± 2.91	0
	500		12.04 ± 49.17	255.2 ± 100.18	1.38 ± 0.01	970.11 ± 378.22	0.2
	750		-	-	-	-	0
	1000		-	-	-	-	0
A_{STD}	1000	100	-919.43 ± 238.36	300.0 ± 0.0	0.46 ± 0.03	1567.29 ± 25.31	0
		1000	-15.92 ± 8.9	300.0 ± 0.0	0.73 ± 0.0	1663.75 ± 8.56	0
		10000	-	-	-	-	0
A_{TE}	25	10000	-479.62 ± 145.46	4.0 ± 0.0	2.31 ± 0.09	14.05 ± 0.25	0
	50		18.31 ± 182.67	3.6 ± 0.55	4.0 ± 0.12	18.72 ± 2.41	0.8
	75		100.0 ± 0.0	4.6 ± 0.89	5.36 ± 0.16	30.01 ± 4.56	1.0
	100		80.0 ± 44.72	4.4 ± 0.89	6.87 ± 0.12	35.74 ± 6.44	0.8

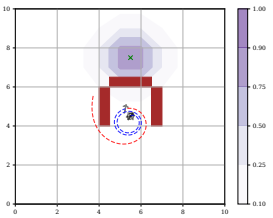
Table A6: Performance of A_{STD} and A_{TE} on the u-shaped map in Dubins Car when the action space is augmented with 100 dummy actions. Both agents use 50 optimization steps. Missing values indicate that the particular configuration was not feasible due to large memory requirements.



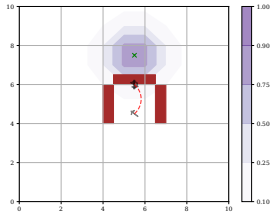
(a) $D_{\text{TE}} = 50, |\mathcal{A}| = 2$



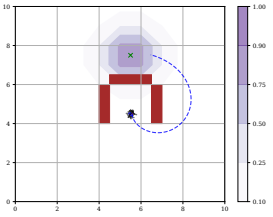
(b) $D_{\text{TE}} = 75, |\mathcal{A}| = 2$



(c) $D_{\text{TE}} = 100, |\mathcal{A}| = 2$



(d) $D_{\text{TE}} = 50, |\mathcal{A}| = 102$



(e) $D_{\text{TE}} = 100, |\mathcal{A}| = 102$

Figure A2: Failure cases for A_{TE} in the u-shaped map. When $|\mathcal{A}| = 2$, for $D_{\text{TE}} = 75$ and $D_{\text{TE}} = 100$, the agent identifies a path around the obstacles but runs out of the maximum number of primitive steps. On increasing the action space, a similar failure occurs for $D_{\text{TE}} = 100$. The failures due to $D_{\text{TE}} = 50$ happen because the agent does not find a path around the obstacles.

Next, we make the problem difficult by augmenting the action space with 100 dummy action variables. As the agents are unaware that these variables don't contribute to the reward or the dynamics, they still need to search over these variables. We set up the experiment similar to the previous one by keeping the number of samples to 10000 and varying the planning horizon. For A_{STD}, almost all the runs fail - either because it does not find a path around the obstacle, or crashes due to huge memory requirements. In order to reduce the memory requirements, we perform a second experiment, fixing the planning horizon to 1000 and varying the number of samples. Reducing the number of samples fixes the issue of memory requirements, but does not help the agent identify a path. In contrast, the performance of A_{TE} is mostly unaffected by this addition of dummy action variables. The detailed results are in Table A6. A_{TE} uses the same configuration as the earlier experiment and achieves similar performance. As observed earlier, the failure case of A_{TE} for planning horizon of 50 is due to the fact that it cannot find a path around the obstacles, while the failure case for planning horizon of 100 is because it runs out of primitive steps.

H Additional MBRL results

H.1 How do the action repeats due to A_{TE} vary across training iterations?

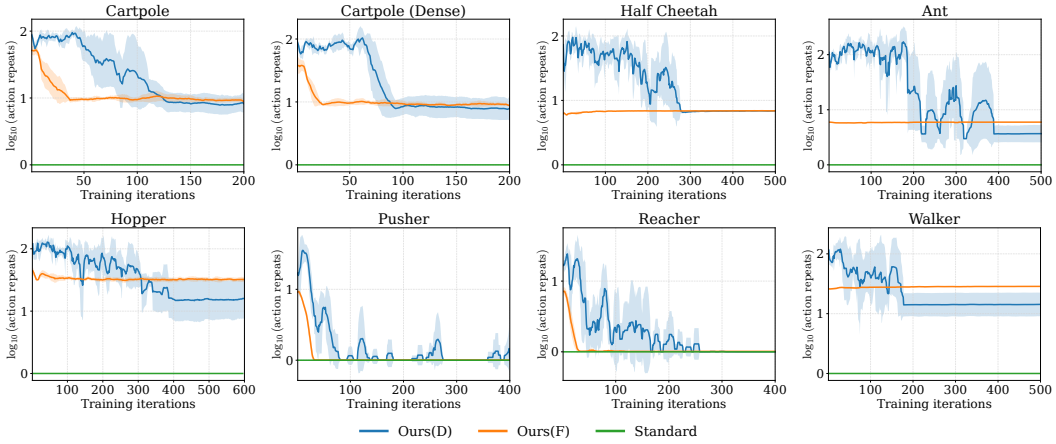


Figure A3: Mean and standard deviation of the running averages (window size=10) of log of action repeats. Values due to $A_{\text{TE}}(\text{D})$ converge in the neighborhood of values due to $A_{\text{TE}}(\text{F})$

.

While the histogram in Figure 4 gives us a good sense of the distribution of discretized action duration $\left(\delta t / \delta t_t^{\text{env}}\right)$ chosen by A_{TE} in one episode after the training has been completed, it does not capture the trend across time. As the average action repeat for $A_{\text{TE}}(\text{D})$ can have a large range, in Figure A3 , we plot the log of average action repeat instead. A_{STD} has an action repeat of 1 as it works with primitive actions. For $A_{\text{TE}}(\text{F})$, the average action repeat is mostly stable. In Cartpole, Reacher and Pusher, the agent has larger action repeats (or takes large values of δt) in the beginning before realizing that smaller action repeats work better. The more interesting plots are those of $A_{\text{TE}}(\text{D})$ as it has to identify the correct range of δt. From the plots, we see that the average action repeats decreases gradually before stabilizing around the average action repeat used by $A_{\text{TE}}(\text{F})$. Note that larger values of average action repeats does not necessarily mean smaller number of decision points. $A_{\text{TE}}(\text{D})$ has to choose $\delta t_{\max}$ candidates from a list of exponentially spaced candidates and a large $\delta t_{\max}$ selection can easily skew the average action repeat.

I Ablations

I.1 Single model vs. Separate models

In non-stationary bandits, the reward distribution of the arms changes with time. If the reward distribution of an arm changes only when it is pulled, it is called a *rested* bandit, while if the reward distribution of an arm changes irrespective of whether that particular arm was pulled or not, it is called a *restless* bandit [Whittle, 1988, Tekin and Liu, 2012].

In the MBRL setup, the performance of the agent depends on the quality of the learned dynamics model. If we learn a single dynamics model to be shared across all arms, then the underlying reward distribution for an arm can keep on changing even if the arm is never pulled. This is the setting of a restless bandit. On the other hand, learning a separate dynamics model for each arm results in a rested bandit setting as it ensures that the underlying reward distribution for an arm only changes when that arm is pulled and the corresponding dynamics gets updated.

Hence, choosing whether to use a single dynamics model which is shared between all the bandit arms or using a separate dynamics model for each bandit is an important consideration. Both the modeling choices have their advantages and disadvantages.

While using a single dynamics makes the most use of the available data, there is a data imbalance issue. Larger values of δt will result in fewer experiences being captured. Even if we uniformly sample the arms, we will have fewer examples due to candidates with large values of $\delta t_{\max}$ than we

will have for candidates with small values of $\delta t_{\max}$. As all of the data is used to train a single model, the model is likely to overfit on examples from candidates with small values of $\delta t_{\max}$. On the other hand, using separate dynamics model for each bandit makes the model more focused, but we do not make full use of the available data.

As shown in Figure A4, we find that using the UCB heuristic, the agent performs better if it learns a separate dynamics model for each bandit arm.

I.2 Which is a better approximation for F - $\hat{F}_{\text{TE}}$ or $\hat{F}_{\text{IP}}$?

While we proposed to use $\hat{F}_{\text{TE}}$ to approximate the temporally-extended dynamics function F, one could also consider using $\hat{F}_{\text{IP}}$ which approximates the one-step dynamics function f and then iterates over it. While we know that $\hat{F}_{\text{TE}}$ leads to better time complexity than $\hat{F}_{\text{IP}}$, it is interesting to explore how it compares in terms of prediction quality.

To answer this question, we perform two experiments with random action sequences and compare the prediction quality of $\hat{F}_{\text{TE}}$ and $\hat{F}_{\text{IP}}$. In both the experiments, we sample random action sequences of length 500 and compute the mean-squared error (MSE) between the actual states observed upon taking the action sequences in the environment and the states as predicted by the learned model. The difference between the two experiments lies in the type of action sequences being sampled. For the first experiment, we randomly sample one-step action sequences which implicitly correspond to duration $\delta t_{\min}$, while for the second experiment, we randomly sample temporally-extended action sequences with action duration sampled from $\mathcal{U}(\delta t_{\min}, \delta t_{\max})$. Each experiment was repeated 10 times in order to get a better estimate and the mean results are shown in Table A7.

			Random one step sequences		Random temporally extended sequences		
Env	$\delta t_{\min}$	$\delta t_{\max}$	$\hat{F}_{\text{TE}}$	$\hat{F}_{\text{IP}}$	$\hat{F}_{\text{TE}}$	$\hat{F}_{\text{IP}}$	$\hat{F}_{\text{IP}}$ without outliers
Cartpole	1	10	0.0107	2.517×10^{-7}	0.0184	0.065	0.065
Ant	0.01	0.07	0.126	0.0149	0.298	4.525×10^{10}	0.60
Half Cheetah	0.01	0.07	0.30	0.04	1.752	2.526	2.526

Table A7: Comparison between the prediction error (in terms of MSE) of $\hat{F}_{\text{TE}}$ and $\hat{F}_{\text{IP}}$ on random action sequences.

In Ant, when predicting using temporally-extended action sequences, in 4 out of 10 experiments, the MSE due to $\hat{F}_{\text{IP}}$ was extremely high. We hypothesize that the states observed during these experiments were probably outside the range of examples seen during the training of the one-step model that is used by $\hat{F}_{\text{IP}}$. $\hat{F}_{\text{IP}}$ without outliers reports the MSE after removing these 4 experiments.

As can be observed, for one-step sequences $\hat{F}_{\text{IP}}$ has a lower MSE but for temporally-extended sequences, $\hat{F}_{\text{TE}}$ is clearly the better choice. This is not surprising since using temporally-extended action sequences with $\hat{F}_{\text{IP}}$ requires iterating over the one-step model multiple times which can lead to compounding errors. Thus, $\hat{F}_{\text{TE}}$ is a better choice both in terms of time complexity and prediction quality when using temporally-extended action sequences.

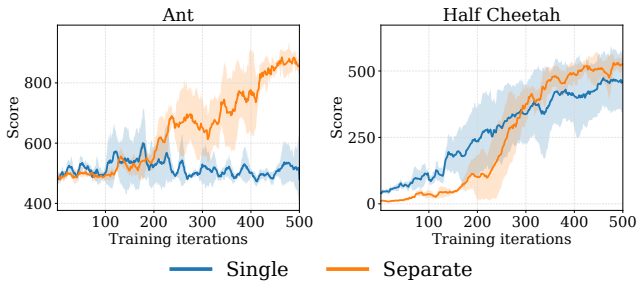


Figure A4: While learning a single dynamics model to be shared across all $\delta t_{\max}$ candidates is sufficient for Half Cheetah, we need to learn separate dynamics models for each $\delta t_{\max}$ candidate in Ant.