A Look at Value-Based Decision-Time vs. Background Planning Methods Across Different Settings

Safa Alver
Mila, McGill University
Montreal, QC, Canada
safa.alver@mail.mcgill.ca

Doina Precup
Mila, McGill University and Google DeepMind
Montreal, QC, Canada
dprecup@cs.mcgill.ca

Abstract

In model-based reinforcement learning (RL), an agent can leverage a learned model to improve its way of behaving in different ways. Two of the prevalent ways to do this are through decision-time and background planning methods. In this study, we are interested in understanding how the value-based versions of these two planning methods will compare against each other across different settings. Towards this goal, we first consider the simplest instantiations of value-based decision-time and background planning methods and provide theoretical results on which one will perform better in the regular RL and transfer learning settings. Then, we consider the modern instantiations of them and provide hypotheses on which one will perform better in the same settings. Finally, we perform illustrative experiments to validate these theoretical results and hypotheses. Overall, our findings suggest that even though value-based versions of the two planning methods perform on par in their simplest instantiations, the modern instantiations of value-based decision-time planning methods can perform on par or better than the modern instantiations of value-based background planning methods in both the regular RL and transfer learning settings.

1 Introduction

It has long been argued that, models will play a key role in building truly intelligent agents that can perform counterfactual reasoning and fast re-planning [Sutton and Barto, 2018, Sutton et al., 2022, Schaul et al., 2018]. Although this is a generally-agreed view in the reinforcement learning (RL) community, the question of how to leverage a model to perform planning does not have a widely-accepted answer. In model-based RL, two of the prevalent ways to perform planning are through decision-time and background planning methods, where the agent mainly plans in the moment and in parallel to its interaction with the environment, respectively [Sutton and Barto, 2018]. Even though these two planning methods have been developed with different scenarios and application domains in mind: (i) decision-time planning algorithms [Tesauro and Galperin, 1996, Silver et al., 2017, 2018] for scenarios in which the exact model of the environment is known and for domains that allow for an adequate computational budget at decision time such as board games like chess and Go, and (ii) background planning algorithms [Sutton, 1990, 1991, Łukasz Kaiser et al., 2020, Hafner et al., 2021, 2023] for scenarios in which the exact model is to be learned through pure interaction with the environment and for domains that are agnostic to the response time of the agent such as gridworlds and video games, recently, with the introduction of the ability to learn a model through pure interaction, decision-time planning algorithms have been applied to the same scenarios and domains as their background planning counterparts: see e.g. Schrittwieser et al. [2020] and Hamrick et al. [2021] which both evaluate a decision-time planning algorithm, called MuZero, on the Atari suite [Bellemare et al., 2013]. However, it still remains unclear in which settings will one of these planning methods perform better than the other.

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

In this study, we attempt to provide an answer to a specific version this question. Specifically, we are interested in the following question:

> *Using the discounted return as the performance measure and considering value-based versions of these methods, in which settings will one planning method perform better than the other?*

To answer this question, we first start by considering the simplest instantiations of value-based decision-time and background planning methods and provide theoretical results on which one will perform better in the regular RL and transfer learning settings. Then, we consider the modern instantiations of them and provide hypotheses on which one will perform better in the same settings. Finally, we perform illustrative experiments to validate these theoretical results and hypotheses. Overall, our results suggest that even though value-based decision-time and background planning methods perform on par in their simplest instantiations, due to (i) not using simulated experience in the updates of the value estimator, and (ii) the ability to improve upon the existing policy in generalization settings, the modern instantiations of value-based decision-time planning methods can perform on par or better than the modern instantiations of value-based background planning methods in both the regular RL and transfer learning settings. We hope that our findings will help the RL community towards developing a better understanding of how the two value-based planning methods will compare against each other across different setting, and thereby stimulate research in improving them in potentially interesting ways.

Key Contributions. The key contributions of this study are as follows: (i) We provide a unified analysis of the value-based versions of the two planning methods and explain how planning is performed within each of these methods (Sec. 2). (ii) We provide theoretical results and hypotheses on how the value-based versions of the two planning methods will compare against each other across different settings (Sec. 3), and (iii) we provide illustrative empirical results that validate these theoretical results and hypotheses (Sec. 4).

2 Background

Reinforcement Learning. In value-based RL [Sutton and Barto, 2018], an agent interacts with its environment through a sequence of actions to maximize its long-term cumulative reward. Here, the environment is usually modeled as a Markov decision process $(\mathcal{S}, \mathcal{A}, p, r, \rho, \gamma)$, where $\mathcal{S}$ and $\mathcal{A}$ are the (finite) set of states and actions, $p : \mathcal{S} \times \mathcal{A} \rightarrow \text{Dist}(\mathcal{S})$ is the transition distribution, $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ is the reward function, $\rho : \mathcal{S} \rightarrow \text{Dist}(\mathcal{S})$ is the initial state distribution, and $\gamma \in [0, 1)$ is the discount factor. The goal of the agent is to learn, through pure interaction, a value estimator $Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ that induces a policy $\pi \in \Pi \equiv \{\pi | \pi : \mathcal{S} \rightarrow \text{Dist}(\mathcal{A})\}$, maximizing $E_{\pi,p}[\sum_{t=0}^{\infty} \gamma^t r(S_t, A_t, S_{t+1}) | S_0 = s_0 \sim \rho]$.

Model-Based RL. One way of achieving this goal is through the use of value-based model-based RL methods, in which there are two alternating phases: the learning and planning phases.[1] In the learning phase, the gathered experience is mainly used in learning a model $m \in \mathcal{M} \equiv \{(p, r, \rho) | p : \mathcal{S} \times \mathcal{A} \rightarrow \text{Dist}(\mathcal{S}), r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}, \rho : \mathcal{S} \rightarrow \text{Dist}(\mathcal{S})\}$, and it may also be used in improving the value estimator. In the planning phase, the learned model m is then used for simulating experience, either to be used alongside real experience in improving the value estimator or just to be used in selecting actions at decision time. In value-based model-based RL methods, similar to value-based RL methods, the agent only employs a value estimator and a model, i.e. it does not employ an explicit policy.

Value-Based Planning Methods. In value-based model-based RL, two of the prevalent ways to perform planning are through value-based decision-time and background planning methods [see Ch. 8 of Sutton and Barto, 2018].[2] For the sake of presentation clarity, in the rest of this study, we will drop the "value-based" adjective and refer to these planning methods as just decision-time and background

[1] We note that even though some model-based RL algorithms, such as AlphaGo/AlphaZero [Silver et al., 2017, 2018], do not employ a model learning phase and make use of a priori given exact model, in this study, we only study algorithms in which the model has to be learned from pure interaction with the environment.

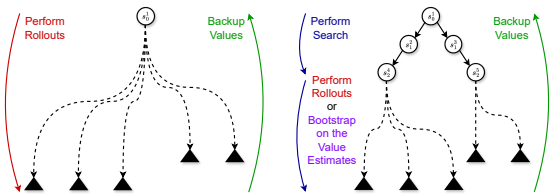
[2] Although some new planning methods have been proposed in the transfer learning literature (see e.g., Barreto et al. [2017, 2019, 2020], Alver and Precup [2022]), these approaches can also be viewed as performing some form of decision-time planning with pre-learned models.

planning methods. In decision-time planning methods, planning is performed as a computation whose output is the selection of a single action for the current state, which is done by unrolling the model forward from the current state to compute local value estimates. Here, planning is performed independently for every encountered state and it is mainly performed in an online fashion, though it may also contain offline components. In contrast, in background planning methods, planning is performed by continually improving a value estimator, on the basis of simulated experience from the model, in a global manner. Action selection is then quickly done by querying the value estimator at the current state. Unlike decision-time planning, background planning is performed in a purely offline fashion, in parallel to the agent-environment interaction, and thus is not focused on the current state. For convenience, in this study, we will refer to all model-based RL algorithms that have an online planning component as decision-time planning algorithms [see e.g. Tesauro and Galperin, 1996, Silver et al., 2017, 2018, Zhao et al., 2021], and will refer to the rest as background planning algorithms [see e.g. Sutton, 1990, 1991, Łukasz Kaiser et al., 2020, Zhao et al., 2021].

It is important to note that in decision-time planning methods, the simulated experience is *only* used for selecting actions at decision time and it is *not* used for updating the value estimator; the value estimator is updated with only *real* experience. In contrast, in background planning methods, the simulated experience is used *alongside* the real experience for updating the value estimator.

Categorization within the Two Planning Methods. Starting with decision-time planning, depending on how much search is performed, decision-time planning methods can be studied under two main categories:

1. Decision-time planning methods that do not perform search
2. Decision-time planning methods that perform search



(a) Pure Rollouts (b) Search + Rollouts/Bootstrap

Figure 1: The different planning methods within decision-time planning in which planning is done (a) by purely performing rollouts, and (b) by first performing some amount of search and then by either performing rollouts or bootstrapping on value estimates. The subscripts and superscripts on the states indicate the time steps and state identifiers, respectively. The black triangles indicate the terminal states.

In the first category, planning is done by just running rollouts (see Fig. 1a). A notable example of a planning method in this category is the online Monte-Carlo planning algorithm of Tesauro and Galperin [1996]. In the second one, planning is done by first performing some amount of search and then either (i) by running rollouts or (ii) by bootstrapping on the value estimates (see Fig. 1b). A notable example of a planning method in this category is the deep model-predictive control algorithm of Zhao et al. [2021]. We refer the reader to Bertsekas [2021] for more details on the different categories within decision-time planning methods.

Moving on to background planning, as all background planning methods perform planning by continually updating a value estimator with simulated experience generated by the model, we do not study them under different categories. Notable examples of background planning methods include the Dyna-Q algorithm of Sutton [1990, 1991] and the deep Dyna-Q algorithm of Zhao et al. [2021].

Value-Equivalent Models. One of the recent trends in model-based RL is to learn models that are specifically useful for planning [Grimm et al., 2020, 2021]. Following Grimm et al. [2021], we define value-equivalent models as follows. Let $V_m^\pi \in \mathbb{R}^{|\mathcal{S}|}$ be the value vector of a policy $\pi \in \Pi$ evaluated in model $m \in \mathcal{M}$, whose elements are defined $\forall s \in \mathcal{S}$ as $V_m^\pi(s) = E_{\pi,p}\left[\sum_{t=0}^{\infty} \gamma^t r(S_t, A_t, S_{t+1}) | S_0 = s\right]$, and let $V_m^* \in \mathbb{R}^{|\mathcal{S}|}$ be the optimal value vector in model m. We say that a model $m_{\text{VE}} \in \mathcal{M}$ is a *value-equivalent (VE) model* of the true environment $m^* \in \mathcal{M}$ if the following equality holds:

$$V_{m^*}^{\pi^*_{m_{\text{VE}}}} = V_{m^*}^* \quad \forall \pi^*_{m_{\text{VE}}} \in \Pi,$$

where $\pi^*_{m_{\text{VE}}}$ is an optimal policy obtained as a result of planning with model m_{VE}.

Evaluation Settings in RL. In RL, two of the most commonly-used evaluation settings are the regular RL and transfer learning settings. In the regular RL setting, the agent is trained and evaluated

on the same task (or distribution of tasks). Although there are many different subcategories within the transfer learning setting [Taylor and Stone, 2009, Zhu et al., 2023], in this study we will only consider the adaptation and generalization settings as these settings are commonly-used for evaluating the simplest and modern instantiations of the two planning styles. In adaptation settings, the agent is first trained and evaluated on a source task (or a distribution of source tasks) and then it is trained and evaluated on a subsequent target task (or a distribution of target tasks). In generalization settings, the agent is just evaluated on a target task (or a distribution of target tasks) as it gets trained and evaluated on a source task (or a distribution of source tasks).

3 Decision-Time vs. Background Planning Methods Across Different Settings

In this study, we are interested in understanding in which settings will one planning method perform better than the other. Thus, we start by formally defining a performance measure that will be used in comparing the two planning methods of interest. Given an arbitrary model $m = (p, r, \rho) \in \mathcal{M}$, we define the performance of an arbitrary policy $\pi \in \Pi$ in it as follows: $J^{\pi}_{m} = E_{\pi,p}[\sum_{t=0}^{\infty} \gamma^t r(S_t, A_t, S_{t+1}) | S_0 = s_0 \sim \rho]$. We note that J^{π}_{m} corresponds to the expected discounted return of a policy π in model m.

3.1 Simplest Instantiations with Tabular Representations

For easy analysis, we start by considering the simplest instantiations of the two planning methods with tabular value estimators and models, which can be found in Ch. 8 of Sutton and Barto [2018]. More specifically, for decision-time planning we study the online Monte-Carlo planning (OMCP) algorithm of Tesauro and Galperin [1996], which falls under the category of decision-time planning algorithms that do not perform search (see Sec. 2), and for background planning we study the Dyna-Q algorithm of Sutton [1990, 1991]. The pseudocode that shows the similarities and differences between these two algorithms is presented in Alg. 1; the only difference between the two algorithms is in how they perform planning (see line 7 for OMCP and lines 10-13 for Dyna-Q), which allows for performing controlled experiments.

Algorithm 1 The joint pseudocode of the OMCP algorithm (with an adaptable model and improving rollout policy) and the Dyna-Q algorithm. The blue and red colored parts are specific to the OMCP and Dyna-Q algorithms, respectively. That is, OMCP does not contain the red parts and Dyna-Q does not contain the blue parts.

1: Initialize Q and m $\forall s \in \mathcal{S}$ and $\forall a \in \mathcal{A}$
2: $n_r \leftarrow$ number of episodes to perform rollouts
3: $n_p \leftarrow$ number of iterations to perform planning
4: **for** each episode **do**
5: $\quad S \leftarrow$ reset environment
6: $\quad$ **while** not done **do**
7: $\quad\quad A \leftarrow \epsilon$-greedy(rollouts$(S, m, n_r, Q)$) / ϵ-greedy$(Q(S, \cdot))$
8: $\quad\quad R, S'$, done $\leftarrow$ environment(A)
9: $\quad\quad$ Update $Q(S, A)$ and $m(S, A)$ with R, S', done
10: $\quad\quad$ **for** $i \in \{1, \ldots, n_p\}$ **do**
11: $\quad\quad\quad S, A \leftarrow$ sample from $\mathcal{S} \times \mathcal{A}$
12: $\quad\quad\quad R, S'$, done $\leftarrow m(S, A)$
13: $\quad\quad\quad$ Update $Q(S, A)$ with R, S', done
14: $\quad\quad S \leftarrow S'$

Regular RL Setting. We start by considering the regular RL setting in which an agent is trained and evaluated on the same task. In this setting, the following statement can easily be proven:

Proposition 1. *Let $m_O \in \mathcal{M}$ and $m_D \in \mathcal{M}$ denote the models of the of the OMCP and Dyna-Q algorithms, respectively, and let π_{m_O} and π_{m_D} denote their final policies generated as a result of planning with their corresponding models. Then, when m_O and m_D become VE models of $m^* \in \mathcal{M}$, we have $J^{\pi_{m_O}}_{m^*} = J^{\pi_{m_D}}_{m^*}$.*

Due to space constraints, we defer the proof to App. A. Prop. 1 states that, the OMCP and Dyna-Q algorithms will perform on par when their models become VE models. More explicitly, in the regular RL setting, we expect the following statement to hold:

Theoretical Result 1. In the regular RL setting, the OMCP algorithm will perform on par with the Dyna-Q algorithm.

Transfer Learning Setting. In the transfer learning setting, we consider an adaptation setting that is commonly-used for evaluating the simplest instantiations of the two planning methods [see e.g. Ch. 8 of Sutton and Barto, 2018]. In this setting, the agent is first trained and evaluated on a source task and then it is trained and evaluated on a subsequent target task. In this setting, Prop. 1 of the regular RL setting will hold directly, as the adaptation setting consists of two consecutive regular RL settings, one with the source task and the other with the subsequent target task. Restating more clearly, in the adaptation setting, we expect the following statement to hold:

> **Theoretical Result 2.** In the adaptation setting, the OMCP algorithm will perform on par with the Dyna-Q algorithm in the source task, and the same will happen in the subsequent target task.

We note that, although we only considered the OMCP and Dyna-Q algorithms, the statements that we provide in this section are generally applicable to most other simplest instantiations of the two value-based planning methods [Sutton and Barto, 2018] as these instantiations also perform planning & learning in a similar fashion described in Alg. 1 and they also use tabular representations in their value estimators and models.

3.2 Modern Instantiations with Neural Network Representations

We now consider the modern instantiations of the two planning methods which represent their models with neural networks. More specifically, for decision-time planning we study the deep model-predictive control (MPC) algorithm of Zhao et al. [2021], which falls under the category of decision-time planning algorithms that perform search (see Sec. 2), and for background planning we study the deep Dyna-Q algorithm of Zhao et al. [2021]. The pseudocode that shows the similarities and differences between these algorithms is presented in Alg. 2. We choose to study these algorithms as the only difference between them is in how they perform planning (see line 12 for deep MPC and lines 17-20 for deep Dyna-Q), that is, similar to the algorithms in Sec. 3.1, these algorithms share the same architectures and learning backbones and only differ in their planning methods, which allows for performing controlled experiments.

Algorithm 2 The joint pseudocode of the deep MPC and deep Dyna-Q algorithms. The blue and red colored parts are specific to the deep MPC and deep Dyna-Q algorithms, respectively. That is, deep MPC does not contain the red parts and deep Dyna-Q does not contain the blue parts. Here, the imaginary replay buffer stores the state-action pairs that are to be used in generating simulated experience.

1: Initialize the parameters θ and ω of Q_θ and m_ω
2: Initialize the replay buffer $\mathcal{D} \leftarrow \{\}$
3: Initialize the imaginary replay buffer $\mathcal{D}_i \leftarrow \{\}$
4: $N \leftarrow$ replay buffer size to start performing updates
5: $n_s \leftarrow$ number of steps to perform search
6: $n_{ib} \leftarrow$ number of samples to sample from $\mathcal{D}_i$
7: $n_b \leftarrow$ number of samples to sample from $\mathcal{D}$
8: $h \leftarrow$ search heuristic
9: **for** each episode **do**
10: $\quad S \leftarrow$ reset environment
11: $\quad$ **while** not done **do**
12: $\quad\quad A \leftarrow \epsilon\text{-greedy}(\text{search+bootstrap}(S, m_\omega, n_s, h, Q_\theta))$ / $\epsilon\text{-greedy}(Q_\theta(S, \cdot))$
13: $\quad\quad R, S', \text{done} \leftarrow \text{environment}(A)$
14: $\quad\quad \mathcal{D} \leftarrow \mathcal{D} + \{(S, A, R, S', \text{done})\}$
15: $\quad\quad \mathcal{D}_i \leftarrow \mathcal{D}_i + \{(S, A)\}$
16: $\quad\quad$ **if** $|\mathcal{D}| \geq N$ **then**
17: $\quad\quad\quad \mathcal{B}_i \leftarrow \text{samplebatch}(\mathcal{D}_i, n_{ib})$
18: $\quad\quad\quad \mathcal{B}_i \leftarrow \mathcal{B}_i + m_\omega(\mathcal{B}_i)$
19: $\quad\quad\quad \mathcal{B} \leftarrow \text{samplebatch}(\mathcal{D}, n_b)$
20: $\quad\quad\quad$ Update Q_θ with $\mathcal{B}$ / $\mathcal{B}_i + \mathcal{B}$
21: $\quad\quad\quad$ Update m_ω with $\mathcal{B}$
22: $\quad\quad S \leftarrow S'$

Regular RL setting. In the regular RL setting, as in the case with the OMCP and Dyna-Q algorithms, we would normally expect the deep MPC and deep Dyna-Q algorithms to perform similarly. However, as neural networks are used in the representations of the value estimators and models, the deep Dyna-Q algorithm can suffer from updating its value estimator with the harmful simulated experiences generated by its pre-VE models, which are earlier versions of the model before it becomes a VE model (see the red part in line 20 of Alg. 2). More specifically, as neural networks are used in the representation of the value estimator, the harmful simulated experiences from the pre-VE models can cause permanent damages in the learning dynamics of the value estimator and slowdown, or prevent, the deep Dyna-Q algorithm in reaching optimal (or good) performance [see e.g. van Hasselt et al., 2019, Jafferjee et al., 2020]. Even though, the pre-VE models of the deep MPC algorithm will also generate harmful simulated experiences,

this will have no permanent effects on its performance, as in deep MPC the value estimator is only updated with real experiences coming from the environment (see the blue part in line 20 of Alg. 2). Thus, we hypothesize that the deep MPC algorithm will perform on par or better than the deep Dyna-Q algorithm, or more explicitly:

> **Hypothesis 1.** In the regular RL setting, as the value estimator of the deep Dyna-Q algorithm will be suffering from the negative effects of the harmful simulated experiences, the deep MPC algorithm will perform on par or better than the deep Dyna-Q algorithm.

Transfer Learning Setting. In the transfer learning setting, we consider adaptation and generalization settings that are commonly-used for evaluating the modern instantiations of the two planning methods [see e.g. Van Seijen et al., 2020, Anand et al., 2022, respectively]. In the adaptation setting, the agent is first trained and evaluated on a distribution of source tasks and then it is trained and evaluated on a distribution of subsequent target tasks. In the generalization setting, the agent is just evaluated on a distribution of target tasks as it gets trained and evaluated on a distribution of source tasks.

In both settings, harmful simulated experiences can again prevent the deep Dyna-Q algorithm from reaching optimal or good performance on the source tasks because of the same reasons discussed in the regular RL setting. Additionally, in the adaptation setting, after the tasks switch from the source tasks to the target tasks, deep Dyna-Q can suffer more in the adaptation process, as its model will keep generating experiences that resembles the source tasks until it adapts to the target tasks, which in the meantime will lead harmful updates to the value estimator. Also, in the generalization setting, if the learned model of the deep MPC algorithm becomes capable of simulating at least a few time steps of the target tasks and if the learned policies of both algorithms perform similarly on the target tasks, the deep MPC algorithm will perform better on the target tasks, as while being evaluated on the target tasks it will be able to improve upon its existing policy via performing online planning; improving upon its existing policy is not possible for the deep Dyna-Q algorithm as it performs planning in an offline fashion and thus requires additional interaction with the target tasks, which is not possible in the generalization settings. More concretely, in the adaptation and generalization settings, we hypothesize that:

> **Hypothesis 2.** In both the adaptation and generalization settings, a similar performance trend with Hypothesis 1 will hold on the source tasks.
>
> **Hypothesis 3.** In the adaptation setting, the deep Dyna-Q algorithm will suffer more in the adaptation process and thus perform worse than the deep MPC algorithm on the target tasks.
>
> **Hypothesis 4.** In the generalization setting, the deep MPC algorithm will improve upon its existing policy and perform better than the deep Dyna-Q algorithm on the target tasks.

We note again that although we only considered the deep MPC and deep Dyna-Q algorithms, the statements that we provide in this section are also generally applicable to most other modern instantiations of the two value-based planning methods [Moerland et al., 2023] as they also perform planning & learning in a similar fashion described in Alg. 2 and also represent their value estimators and models with neural networks.

4 Experimental Results

We now perform illustrative experiments to validate the theoretical and hypothetical statements presented in Sec. 3.

Environmental Details. As a testbed, we use (i) the Simple Gridworld environment (see Fig. 2a), and (ii) five environments from MiniGrid [see Fig. 2b-2h, Chevalier-Boisvert et al., 2018]. We choose these environments as VE models are easy to learn in them and thus they allow for designing controlled experiments that are helpful in answering the questions of interest to this study.

In the Simple Gridworld (SG) environment, the agent spawns in state S and has to navigate to the goal state G. At each time step, the agent receives coordinates indicating its position, and based on this selects an action that moves it to one of the four neighboring cells. The agent receives a reward of $+1$ if it reaches G and the episode terminates. In MiniGrid environments, the agent (depicted in red) has to navigate to the green goal cell, while avoiding the orange lava cells (if there are any). At each

time step, the agent receives a top-down image of the grid and based on this chooses an action that moves it to one of the four neighboring cells. If the agent steps on a lava cell, the episode terminates with no reward, and if it reaches the goal cell, the episode terminates with a reward of $+1$.

4.1 Experiments with Simplest Instantiations

In this section, we perform experiments with the OMCP algorithm of Tesauro and Galperin [1996] and the Dyna-Q algorithm of Sutton [1990, 1991] on the SG environment to illustrate our theoretical results presented in Sec. 3.1. The implementation details of these algorithms can be found in App. B.1.

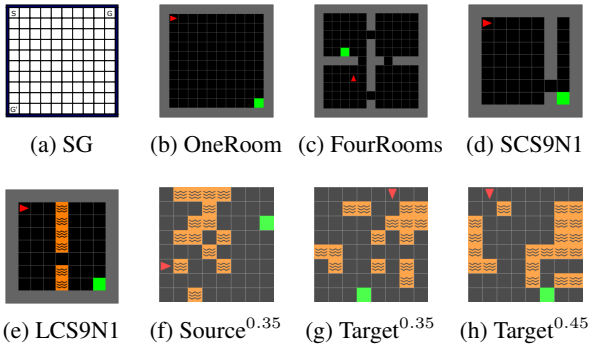

(a) SG (b) OneRoom (c) FourRooms (d) SCS9N1
(e) LCS9N1 (f) Source$^{0.35}$ (g) Target$^{0.35}$ (h) Target$^{0.45}$

Figure 2: (a) The SG environment. (b-e) The OneRoom, FourRooms, SCS9N1 and LCS9N1 environments in MiniGrid. (f-h) The source task of difficulty 0.35 and target tasks of difficulties 0.35 and 0.45 in the RDS environment. The difficulty parameter here controls the density of the lava cells between the agent and the goal cell, and that the target tasks are just transposed versions of the source task. With every reset of the episode, a new lava cell pattern is procedurally generated for both the source and target tasks.

Regular RL Setting. Theoretical Result 1 states that, in the regular RL setting, the OMCP algorithm will perform on par with the Dyna-Q algorithm. For empirical illustration, we compared these algorithms on the SG environment. The empirical result in Fig. 3a shows that, as expected, the OMCP algorithm performs on par with the Dyna-Q algorithm, which illustrates Theoretical Result 1.

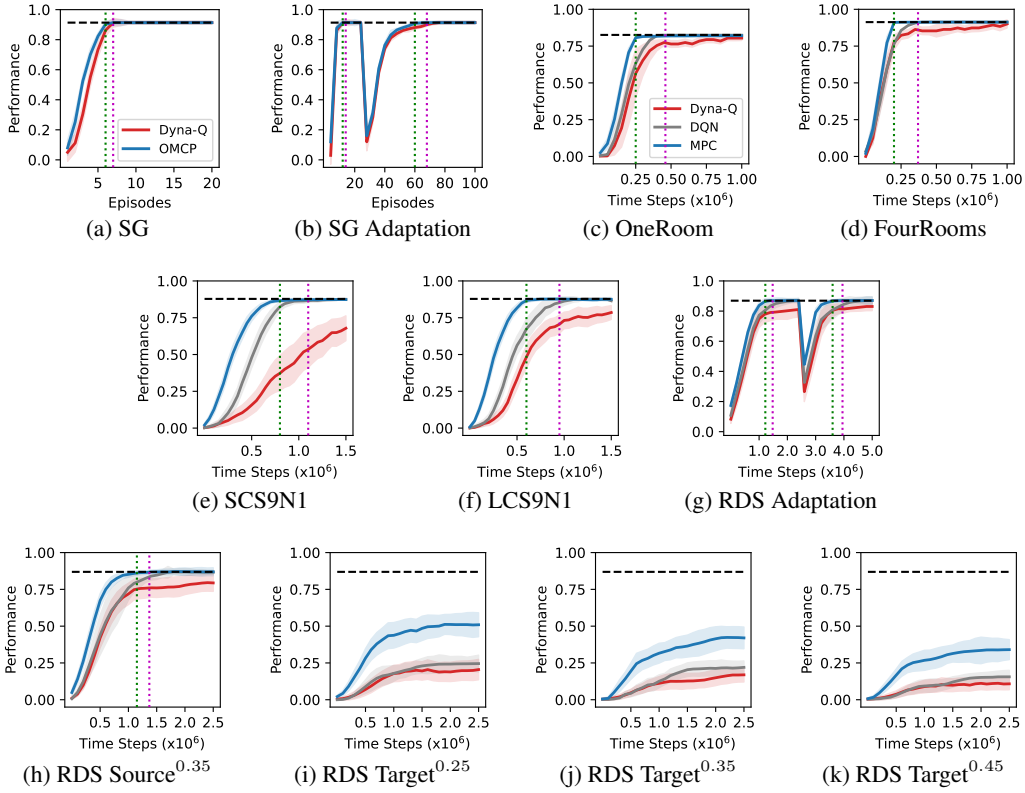
Transfer Learning Setting. Theoretical Result 2 states that in the adaptation setting, the OMCP algorithm will perform on par with the Dyna-Q algorithm in the source task, and the same will happen in the subsequent target task. For illustration, we compared these algorithms on an adaptation version of the SG environment. In this environment, the agent is first trained and evaluated on a source task with goal G and then it is subsequently trained and evaluated on a target task with with goal G$^{'}$ (see Fig. 2a). In Fig. 3b, we can see that, the OMCP algorithm first performs on par with the Dyna-Q algorithm in the source task and the same happens in the target task, illustrating Theoretical Result 2.

4.2 Experiments with Modern Instantiations

We now perform experiments with the deep MPC and deep Dyna-Q algorithms of Zhao et al. [2021] on MiniGrid environments to empirically validate our hypotheses in Sec. 3.2. In addition to these two algorithms, we also perform experiments with the model-free DQN algorithm [Mnih et al., 2015] to illustrate (i) the negative effects of the usage of simulated experiences in the deep Dyna-Q algorithm and (ii) the positive effects of performing online planning in the deep MPC algorithm. We note that the only difference between deep Dyna-Q and DQN is that in the DQN algorithm the value estimator is only updated with real experiences (see line 17-20 in Alg. 1). We also note that the only difference between deep MPC and DQN is that in the DQN algorithm there is no online planning component (see line 12 in Alg. 2). The implementation details of these algorithms can be found in App. B.2.

Regular RL Setting. In Hypothesis 1, we hypothesized that in the regular RL setting, the deep MPC algorithm will perform on par or better than the deep Dyna-Q algorithm as the value estimator of the deep Dyna-Q algorithm will suffer from the negative effects of the harmful simulated experiences. To validate this hypotheses, we compared these algorithms on four MiniGrid environments: OneRoom, FourRooms, SimpleCrossingS9N1 (SCS9N1) and LavaCrossingS9N1 (LCS9N1). The results, in Fig. 3c-3f, show that the deep MPC algorithm indeed performs on par or better than deep Dyna-Q algorithm, even after deep Dyna-Q's model becomes a VE model, which validates Hypothesis 1.

To make sure that the simulated experiences are causing the problem, we also trained the DQN algorithm on these environments. Results in Fig. 3c-3f, show that the DQN algorithm achieves the same performance with the deep MPC algorithm, illustrating the negative effects of using simulated experience in the updates of the value estimator in the deep Dyna-Q algorithm.



(a) SG (b) SG Adaptation (c) OneRoom (d) FourRooms

(e) SCS9N1 (f) LCS9N1 (g) RDS Adaptation

(h) RDS Source$^{0.35}$ (i) RDS Target$^{0.25}$ (j) RDS Target$^{0.35}$ (k) RDS Target$^{0.45}$

Figure 3: (a, b) The performance of OMCP and Dyna-Q in the (a) regular RL and (b) transfer learning settings. (c-k) The performance of deep MPC, deep Dyna-Q (abbreviated as MPC and Dyna-Q) and DQN in the (c-f) regular RL and (g-k) transfer learning settings. The black dashed lines indicate the performance of the optimal policy in the corresponding environment. The green and magenta dotted lines indicate the point after which the models of the decision-time and background planning algorithms become and remain as VE models, respectively. The shaded regions are confidence intervals over (a, b) 50 and (c-k) 25 runs.

Transfer Learning Setting. In Sec. 3.2, we first hypothesized in Hypothesis 2 that in both the adaptation and generalization settings, a similar performance trend with the regular RL setting will hold on the source tasks. Then, we hypothesized in Hypothesis 3 that in the adaptation setting, the deep Dyna-Q algorithm will suffer more in adaptation process and thus perform worse than the deep MPC algorithm on the target tasks. Finally, we hypothesized in Hypothesis 4 that in the generalization setting, the deep MPC algorithm will improve upon its existing policy and perform better than the deep Dyna-Q algorithm on the target tasks.

In order to test the validity of Hypothesis 2 and 3, which are related to the adaptation setting, we compared the deep MPC and deep Dyna-Q algorithms on an adaptation version of the RandDistShift [RDS, Zhao et al., 2021] environment. In this environment, the agent is first trained and evaluated on source tasks with difficulty 0.35 (see Fig. 2f) and then it is subsequently trained and evaluated on target tasks with difficulty 0.35 (see Fig. 2g). Results in Fig. 3g show that (i) the deep Dyna-Q algorithm indeed performs worse than the deep MPC algorithm on the source tasks, and (ii) it indeed suffers more in the adaptation process and performs worse than deep MPC on the target tasks, validating Hypothesis 2 and 3, respectively.

In order to test the validity of Hypothesis 2 and 4, which are related to the generalization setting, we compared the deep MPC and deep Dyna-Q algorithms on the original RDS environment [Zhao et al., 2021]. In this environment, the agent is trained and evaluated on source tasks with difficulty 0.35 (see Fig. 2f) and during the this process it is periodically evaluated on the target tasks with difficulties varying from 0.25 to 0.45 (see Fig. 2g-2h). Results are shown in Fig. 3h-3k. As can be seen in Fig. 3h, the deep Dyna-Q algorithm again performs worse than the deep MPC algorithm on the source

tasks, further validating Hypothesis 2. We can also see in Fig. 3i-3k that deep MPC indeed achieves significantly better generalization performance than deep Dyna-Q across all target tasks with varying difficulties, validating Hypothesis 4.

To make sure that the simulated experiences are causing the problem in the deep Dyna-Q algorithm and that the online planning component is leading to a generalization performance improvement in the deep MPC algorithm, we also trained the DQN algorithm on the RDS environments. Results in Fig. 3g-3h, show that the DQN algorithm achieves the same performance with the deep MPC algorithm, again illustrating the negative effects of using simulated experience in the updates of the value estimator in the deep Dyna-Q algorithm. We can also see in Fig. 3i-3k that performing online planning significantly boosts the performance of the deep MPC algorithm compared to the DQN algorithm.

5 Related Work

In the model-based RL literature, there have been benchmarking studies that empirically compare the performances of various decision-time and background planning algorithms on continuous control domains in the regular RL setting [Wang et al., 2019] and on MiniGrid environments in specific transfer learning settings [Zhao et al., 2021]. However, unlike our study, none of these studies provide a unified analysis of when will one planning method perform better than the other across different settings. Also, rather than comparing the algorithms using the expected discounted return, these studies perform the comparison using the expected undiscounted return, and thus might be misleading in understanding the degree of optimality of the generated output policies.

The study of Moerland et al. [2023] reviews many of the decision-time and background planning algorithms proposed in the model-based RL literature. However, it does not focus on providing an answer to the question of how would the two prevalent planning methods in model-based RL compare against each other across different settings; in contrast, this study exactly focuses on this question.

The categorization within the decision-time planning algorithms that we provide in this study (see Sec. 2) has connections to the recent monograph of Bertsekas [2021] in which the recent successes of AlphaZero [Silver et al., 2018], a decision-time planning algorithm, are viewed through the lens of dynamic programming. However, we take a broader perspective and provide a categorization that encompasses both decision-time and background planning algorithms. Also, in this study, instead of assuming the availability of an exact model, we consider scenarios in which a model has to be learned by pure interaction with the environment.

Another related study is the study of Hamrick et al. [2021] which aims for providing an understanding of how the different components of decision-time algorithms can affect their performance across different settings by studying the MuZero algorithm [Schrittwieser et al., 2020]. Rather than just considering decision-time planning algorithms, in this study we also consider background planning algorithms and we provide a discussion on how one of the key characteristics of background planning can hinder its performance across different settings.

6 Conclusion and Discussion

To summarize, we performed a unified analysis of value-based decision-time and background planning methods and attempted to answer the following question: *using the discounted return as the performance measure and considering value-based versions of these methods, in which settings will one planning method perform better than the other?* In our analysis, we have studied generic algorithms and provided theoretical and hypothetical statements that are generally applicable to most other instantiations of the two value-based planning methods in their corresponding classes. Overall, our findings suggest that even though value-based decision-time and background planning methods perform on par on their simplest instantiations, the modern instantiations of value-based decision-time planning methods can perform on par or better than the modern instantiations of value-based background planning methods in both the regular RL and transfer learning settings.

We note that the main purpose of this study was to contribute towards the goal of providing a general understanding of in which settings will one value-based planning method perform better than the other through studying the generic algorithms in their corresponding classes, and not to provide a

benchmark that compares state-of-the-art model-based RL algorithms across various settings and domains. We also note that even though providing practical insights is not the main goal of this study, we believe that our study can guide the community in improving value-based background planning methods in potentially interesting ways. For example, a possible improvement to modern background planning algorithms could be to add a meta-level algorithm that controls the amount of simulated data in the updates of the value estimator throughout learning. Finally, we note that we were only interested in comparing the two planning methods in terms of the expected discounted return of their output policies; though not the main focus of this study, other possible interesting comparison directions include comparing the two methods in terms of their sample efficiency and real-time performance.

References

Safa Alver and Doina Precup. Constructing a good behavior basis for transfer using generalized policy updates. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=7IWGzQ6gZ1D`.

Ankesh Anand, Jacob C Walker, Yazhe Li, Eszter Vértes, Julian Schrittwieser, Sherjil Ozair, Theophane Weber, and Jessica B Hamrick. Procedural generalization by planning with self-supervised world models. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=FmBegXJToY`.

André Barreto, Will Dabney, Rémi Munos, Jonathan J Hunt, Tom Schaul, Hado P van Hasselt, and David Silver. Successor features for transfer in reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

André Barreto, Diana Borsa, Shaobo Hou, Gheorghe Comanici, Eser Aygün, Philippe Hamel, Daniel Toyama, Jonathan Hunt, Shibl Mourad, David Silver, et al. The option keyboard combining skills in reinforcement learning. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pages 13052–13062, 2019.

André Barreto, Shaobo Hou, Diana Borsa, David Silver, and Doina Precup. Fast reinforcement learning with generalized policy updates. *Proceedings of the National Academy of Sciences*, 117 (48):30079–30087, 2020.

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47: 253–279, 2013.

Dimitri P Bertsekas. *Rollout, policy iteration, and distributed reinforcement learning*. Athena Scientific, 2021.

Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for openai gym. `https://github.com/maximecb/gym-minigrid`, 2018.

Christopher Grimm, Andre Barreto, Satinder Singh, and David Silver. The value equivalence principle for model-based reinforcement learning. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 5541–5552. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper/2020/file/3bb585ea00014b0e3ebe4c6dd165a358-Paper.pdf`.

Christopher Grimm, Andre Barreto, Gregory Farquhar, David Silver, and Satinder Singh. Proper value equivalence. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL `https://openreview.net/forum?id=aXbuWbta0V8`.

Danijar Hafner, Timothy P Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=0oabwyZbOu`.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023.

Jessica B Hamrick, Abram L. Friesen, Feryal Behbahani, Arthur Guez, Fabio Viola, Sims Witherspoon, Thomas Anthony, Lars Holger Buesing, Petar Veličković, and Theophane Weber. On the role of planning in model-based deep reinforcement learning. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=IrM64DGB21`.

Taher Jafferjee, Ehsan Imani, Erin Talvitie, Martha White, and Micheal Bowling. Hallucinating value: A pitfall of dyna-style planning with imperfect environment models. *arXiv preprint arXiv:2006.04363*, 2020.

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.

Thomas M Moerland, Joost Broekens, Aske Plaat, Catholijn M Jonker, et al. Model-based reinforcement learning: A survey. *Foundations and Trends® in Machine Learning*, 16(1):1–118, 2023.

Tom Schaul, Hado van Hasselt, Joseph Modayil, Martha White, Adam White, Pierre-Luc Bacon, Jean Harb, Shibl Mourad, Marc Bellemare, and Doina Precup. The barbados 2018 list of open issues in continual learning. *arXiv preprint arXiv:1811.07004*, 2018.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419): 1140–1144, 2018.

Richard S Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine learning proceedings 1990*, pages 216–224. Elsevier, 1990.

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

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Richard S Sutton, Michael H Bowling, and Patrick M Pilarski. The alberta plan for ai research. *arXiv preprint arXiv:2208.11173*, 2022.

Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(7), 2009.

Gerald Tesauro and Gregory Galperin. On-line policy improvement using monte-carlo search. *Advances in Neural Information Processing Systems*, 9:1068–1074, 1996.

Hado P van Hasselt, Matteo Hessel, and John Aslanides. When to use parametric models in reinforcement learning? *Advances in Neural Information Processing Systems*, 32:14322–14333, 2019.

Harm Van Seijen, Hadi Nekoei, Evan Racah, and Sarath Chandar. The loca regret: A consistent metric to evaluate model-based behavior in reinforcement learning. *Advances in Neural Information Processing Systems*, 33:6562–6572, 2020.

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.

Mingde Zhao, Zhen Liu, Sitao Luan, Shuyuan Zhang, Doina Precup, and Yoshua Bengio. A consciousness-inspired planning agent for model-based reinforcement learning. *Advances in Neural Information Processing Systems*, 34, 2021.

Zhuangdi Zhu, Kaixiang Lin, Anil K Jain, and Jiayu Zhou. Transfer learning in deep reinforcement learning: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.

Łukasz Kaiser, Mohammad Babaeizadeh, Piotr Miłos, Błażej Osiński, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, Afroz Mohiuddin, Ryan Sepassi, George Tucker, and Henryk Michalewski. Model based reinforcement learning for atari. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=S1xCPJHtDB`.

A Proofs

Proposition 1. *Let $m_O \in \mathcal{M}$ and $m_D \in \mathcal{M}$ denote the models of the of the OMCP and Dyna-Q algorithms, respectively, and let π_{m_O} and π_{m_D} denote their final policies generated as a result of planning with their corresponding models. Then, when m_O and m_D become VE models of $m^* \in \mathcal{M}$, we have $J_{m^*}^{\pi_{m_O}} = J_{m^*}^{\pi_{m_D}}$.*

Proof. This result directly follows from the definition of VE models (see Sec. 2). Recall that when a model $m \in \mathcal{M}$ is a VE model, the following equality holds:

$$V_{m^*}^{\pi_m^*} = V_{m^*}^* \quad \forall \pi_m^* \in \Pi,$$

which implies $V_{m^*}^{\pi_{m_O}^*} = V_{m^*}^{\pi_{m_D}^*} = V_{m^*}^*$ for all $\pi_{m_O}^*, \pi_{m_D}^* \in \Pi$. This in turn implies $J_{m^*}^{\pi_{m_O}} = J_{m^*}^{\pi_{m_D}}$. □

B Implementation Details

In this section, we provide the implementation details of the simplest and modern instantiations of the two planning methods that were considered in this study.

B.1 Implementation Details of the Simplest Instantiations

As stated in the main text, for our simplest instantiation experiments, we considered the OMCP algorithm of Tesauro and Galperin [1996] and the Dyna-Q algorithm of Sutton [1990, 1991]. The joint pseudocode that shows the similarities and differences between these two algorithms is presented in Alg. 1. The details and hyperparameters of these algorithms are presented in Table B.1.

Table B.1: Details and hyperparameters of the OMCP and Dyna-Q algorithms presented in Alg. 1. The blue and red colored parts are specific to the OMCP and Dyna-Q algorithms, respectively.

Details and hyperparameters	Values
representation of Q	tabular value function with $\lvert\mathcal{S}\rvert \times \lvert\mathcal{A}\rvert$ entries
representation of m	tabular model with $\lvert\mathcal{S}\rvert \times \lvert\mathcal{A}\rvert$ entries
number of episodes to perform to perform rollouts, n_r	50
number of iterations to perform planning, n_p	100
exploration rate, ϵ	linearly decays from 1.0 to 0.0 over 5 episodes
learning rate	1e-1
discount factor	0.95

Both of these algorithms use the update rule of tabular Q-learning for updating $Q(S, A)$:

$$Q(S, A) \leftarrow Q(S, A) + \alpha \left[R + \gamma \max_{a' \in \mathcal{A}} Q(S', a') - Q(S, A) \right],$$

and a regular update rule from supervised learning for updating $m(S, A)$:

$$m(S, A) \leftarrow S', R, \text{done}.$$

B.2 Implementation Details of the Modern Instantiations

As stated in the main text, for our modern instantiation experiments, we considered the deep MPC and deep Dyna-Q algorithms in Zhao et al. [2021], which are referred to as "MPC (UP)" and "Dyna" in Zhao et al. [2021]. The joint pseudocode that shows the similarities and differences between these two algorithms is presented in Alg. 2. The details and hyperparameters of these algorithms are presented in Table B.3.

Table B.2: The neural network architectures of the value estimators and models of the deep MPC and deep Dyna-Q algorithms.

Components	Value estimator	CNN: (Channels: $[32 \times 64 \times 64]$ Kernel Sizes: $[8 \times 3 \times 3]$ Strides: $[4 \times 2 \times 2]$), Followed by MLP: $[512]$, Activation Function: ReLU
	Dynamics model	CNN: (Channels: $[32 \times 64 \times 64]$ Kernel Sizes: $[8 \times 3 \times 3]$ Strides: $[4 \times 2 \times 2]$), Followed by Transposed CNN: (Channels: $[64 \times 32 \times 3]$ Kernel Sizes: $[6 \times 6 \times 5]$ Strides: $[1 \times 4 \times 3]$), Activation Function: ReLU
	Reward model	CNN: (Channels: $[32 \times 64 \times 64]$ Kernel Sizes: $[8 \times 3 \times 3]$ Strides: $[4 \times 2 \times 2]$), Followed by MLP: $[512]$, Activation Function: ReLU
	Termination model	CNN: (Channels: $[32 \times 64 \times 64]$ Kernel Sizes: $[8 \times 3 \times 3]$ Strides: $[4 \times 2 \times 2]$), Followed by MLP: $[512]$, Activation Function: ReLU

Table B.3: Details and hyperparameters of the deep MPC and deep Dyna-Q algorithms presented in Alg. 2. The blue and red colored parts are specific to the deep MPC and deep Dyna-Q algorithms, respectively.

Details and hyperparameters	**Values**
representation of Q_θ	CNN with an architecture as in Table B.2
representation of m_ω	CNN with an architecture as in Table B.2
size of $\mathcal{D}$	1e6
size of $\mathcal{D}_i$	1e6
replay buffer size to start updates, N	5e4
number of steps to perform search, n_s	5
number of samples to sample from $\mathcal{D}_i$, n_{ib}	128
number of samples to sample from $\mathcal{D}$, n_b	128
search heuristic, h	best-first search
exploration rate, ϵ	linearly decays from 1.0 to 0.0 over 1M time steps
learning rate	1e-4
optimizer type	Adam
discount factor	0.95

In both of these algorithms, the parameters of the value estimator Q_θ are updated towards minimizing the TD loss $\mathcal{L}_{\text{TD}}(\theta)$, which is calculated in a similar fashion to DQN [Mnih et al., 2015]; and the parameters of the model m_ω are updated towards minimizing the dynamics loss $\mathcal{L}_{\text{dyn}}(\omega)$, reward loss $\mathcal{L}_{\text{r}}(\omega)$ and termination loss $\mathcal{L}_{\text{t}}(\omega)$. Specifically, $\mathcal{L}_{\text{dyn}}(\omega)$ is the ℓ_2 distance between the between the predicted next state $\hat{S}'$ and the true next state S', $\mathcal{L}_{\text{r}}(\omega)$ is the ℓ_2 distance between the between the predicted reward $\hat{R}$ and the true reward R, and $\mathcal{L}_{\text{t}}(\omega)$ is the binary cross-entropy loss between the imagined termination and the true termination. For more on the training details of the value estimators and models of the two algorithms, we refer the reader to Zhao et al. [2021] and its publicly available code[3].

[3]See `https://github.com/mila-iqia/Conscious-Planning`.

In our modern instantiation experiments, we also considered the DQN algorithm [Mnih et al., 2015]. This algorithm is just a version of the deep Dyna-Q algorithm in which real experiences are used in place of the simulated experiences for updating the value estimator. Its details and hyperparameters are the same with the deep Dyna-Q algorithm.

Additionally, we note that in our adaptation experiments, we reinitialized the replay buffers of all algorithms after the tasks switched from the source tasks to the target tasks.