# MULTI-MODEL REINFORCEMENT LEARNING WITH ONLINE RETROSPECTIVE CHANGE-POINT DETECTION

**Augustin Chartouny**[1*] **Mehdi Khamassi**[1] **Benoît Girard**[1]

[1]Institut des Systèmes Intelligents et de Robotique, UMR 7222

Sorbonne Université - CNRS

Paris, France

**ABSTRACT**

Unlike humans, who continuously adapt to both known and unknown situations, most reinforcement learning agents struggle to adjust to changing environments. In this paper, we present a new model-based reinforcement learning method that adapts to local task changes online. This method can detect changes at the level of the state-action pair, create new models, retrospectively update its models depending on when it estimates that the change happened, reuse past models, merge models if they become similar, and forget unused models. This new change detection method can be applied to any Markov decision process, to detect changes in transitions or in rewards. Arbitrating between local models limits memory costs and enables faster adaptation to new contexts which sub-parts have been experienced before. We provide a thorough analysis of each parameter of the multi-model agent and demonstrate that the multi-model agent is stable and outperforms single-model methods in uncertain environments. This novel multi-model agent yields new insights and predictions concerning optimal decision-making in changing environments, which could in turn be tested by future experiments in humans.

***Keywords*** Model-based reinforcement learning · Non-stationarity · Model switching · Task change · Change detection

## 1 Introduction

Humans constantly adapt to uncertain, new, and open-ended environments in their daily lives. To explain human behavioral flexibility in changing environments, researchers study contextual inference and reinforcement learning. Contextual inference refers to online interpretation of environmental cues to discriminate between different situations [21]. Reinforcement learning, on the other hand, explores how humans refine their strategies over time to make optimal decisions based on their interactions with an environment [26]. Modeling how humans infer contexts and adapt their behavior online is essential to understand human behavior flexibility.

Nevertheless, traditional approaches to contextual inference study task changes at the whole-task level [8, 3, 14]. When detecting a change, these traditional context-level approaches change all of their models of the environment simultaneously. This poses a number of limitations. Firstly, the assumption that humans learn to differentiate task changes solely at the context level would not explain the facilitated transfer of knowledge between tasks that share common properties. This assumption suggests that agents relearn an entire context to adapt even for minor variations, which would be inefficient for learning new contexts [17]. For instance, a rat exploring a maze is not obligated to relearn the entire structure of the maze every time a door is opened or closed [20]. As the context only changes locally, the rat is only required to learn two models for the door, and can use a shared model for the rest of the maze.

Secondly, detecting a task change at the context-level relies on aggregate measures such as cumulative sums of local changes [14]. Thus, contextual inference methods often have trouble identifying the exact factors that drive contextual

*Contact: augustin.chartouny[at]sorbonne-universite.fr

shifts, including the precise moment or location of each change. This, added with high computational costs due to Bayesian change detection methods [13], makes it impossible to retrospectively assign observations to the correct context after a change has been identified. As a result, context-level approaches tend to assume that a change occurs just before or at the exact point of detection [8, 10, 14]. In contrast, humans often make sense of a detected change by retrospectively inferring a latent cause [25, 22]. For instance, let's take the example of someone trying to guess when a fair coin is replaced by a rigged one, using the sequence of coin tosses only. They need several observations to detect a change but once they detect a change, they do not simply assume that the change occurred at the very moment they noticed it. Rather, they can try to retrospectively infer the moment that the fair coin was replaced by the rigged one based on the toss distribution. Context-level methods that lack the ability to retrospectively assign observations to the correct context fail to capture this fundamental aspect of task change detection in humans.

To address these limitations, we propose a novel approach that operates at the local level, utilizing multiple local models and only modifying local models which change between contexts. By detecting changes at the local level, this method enhances the transfer of knowledge between contexts and improves learning, using retrospective change-point detection.

### 1.1 Context-level and multi-model approaches

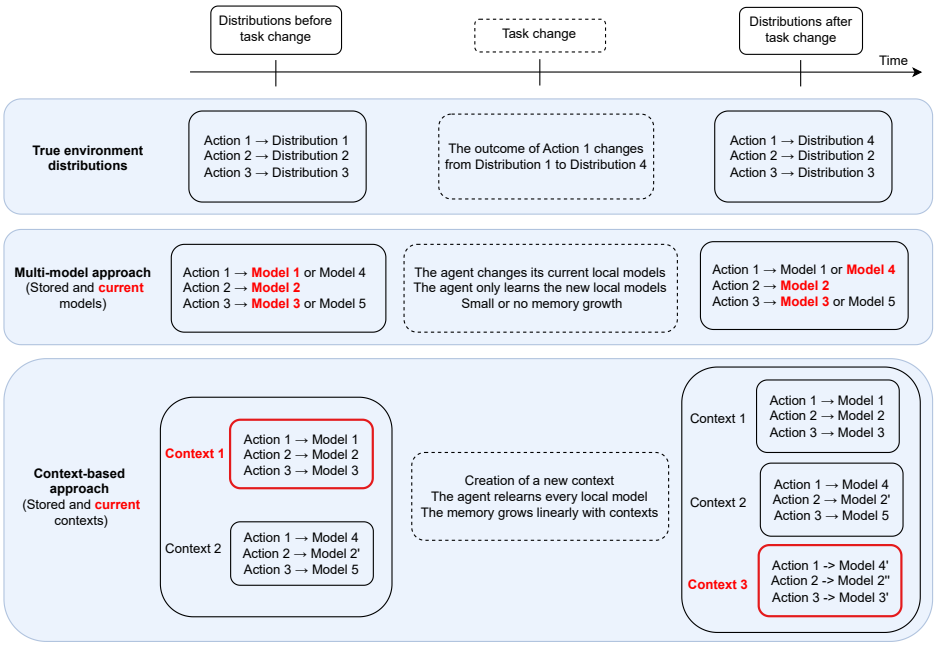


Figure 1: **Comparison between context-level and multi-model approaches in a three-action task.** Both approaches model the result of each action, for example the distribution of rewards in a three-armed bandit. If contexts share features, arbitrating directly between models is theoretically more efficient than arbitrating between contexts. The task change involves a change in the result of action 1 from distribution 1 to distribution 4. The context-level approach must create a new context and relearn a model for each action. The models the context-level agent relearn are indicated with apostrophes. For example, models 2, 2′', and 2′′ are three different models of distribution 2. Conversely, the multi-model approach only relearns the local models that changed during the task change. Red text indicates the best context or the best models for each method based on the true environment distributions.

The definition of contexts varies in decision-making and reinforcement learning. Contexts are generally considered to be an ensemble of properties of the environment that an agent must perceive to behave efficiently [16]. We consider here that a context is a mode of a Markov Decision Process, using the formalism of Hidden-Mode Markov Decision Processes [6]. The environment the agent interacts with can be in different stationary Markov Decision Processes, namely modes or contexts, and the agent has no control over when the environment changes from one context to another. Each context is likely to remain stable for some time, and we make no assumptions regarding the total number of contexts an agent may face. This is similar to the unbounded number of latent causes often found in the Bayesian literature [13]. For instance, the changing environment may be a non-stationary three-armed bandit task [27, 7]. For a three-armed bandit, a context would be comprised of three reward distributions – one for each of the three arms

(Figure 1). The context changes when the distribution of one or several arms change. An agent must model the three reward distributions and adapt appropriately to context changes to perform optimally.

Context-level methods modify all of the local models learned when detecting a context change [10, 14, 8]. As mentioned above, this grants prompt adaptation to changes only after an agent has learned all possible contexts, thus preventing the leveraging of cross-context similarities when a new context is being learned. Moreover, memory usage increases linearly as a function of the number of contexts, irrespective of cross-context similarities. Context-level approaches are also inefficient in detecting task changes when an environment is only modified locally, due to the high degree of similarity between contexts. Subsequently, if local changes occur frequently, context-level approaches may fail to adapt or require full context updates consistently. Both of these issues would give way too high computational costs.

Considering the above-outlined disadvantages of context-level approaches, we believe that a local-model-level approach would be more appropriate for contexts that share features. We propose an alternative approach that detects changes in local models (as opposed to entire contexts), adapting to them accordingly (Figure 1, right). Not only does our approach reduce memory usage and computational costs, but it also favors transfer learning; Only cross-context differences are learned and stored. The local-model-level approach encourages adaptation to local changes and promotes the transfer of prior knowledge to new contexts. However, one disadvantage of our approach is that, in contexts that are dissimilar and familiar, it may detect changes slower than context-level methods.

### 1.2 retrospective change-point detection

Detecting changes at the local level facilitates finding the source of surprising events, which come from either the uncertainty of a stable environment or a change in the environment. To categorize surprising events from an observer's perspective, the decision-making literature often disentangles expected uncertainty from unexpected uncertainty [1]. Expected uncertainty consists of the predicted variability of outcomes in a stable environment. Unexpected uncertainty consists of events that violate model expectations and can come from a change in the environment. Hence, decision-makers who face a surprising event must assess to what extent their model explains it. When their current model does not represent the current situation anymore, they must change their model of the environment.

However, one surprising event may not be enough to assess the reliability of a model. For example, when trying to understand whether a die is fair or rigged, observing one roll with an outcome between 1 and 6 is not very informative. However, observing only sixes for the first five rolls may indicate that the die is rigged. To further understand how to detect a change in a distribution, we consider the toy problem of a passive observer trying to determine whether a die is rigged or fair. The observer only has access to the results of the rolls. Nevertheless, they know that there are only two dice, one fair and one rigged. Moreover, they understand that the chance to switch from one die to another is small after a die change and that each die is rolled several times in a row. The roll sequence is the following, with labels indicating rigged or fair die for the reader:

$$\{\underbrace{6,5,2,4,4}_{\text{fair die}},\underbrace{3,3,2,2}_{\text{rigged die}},\underbrace{5,1,6,4,2,4,1}_{\text{fair die}},\underbrace{2,2,3,2,3,2,2,3}_{\text{rigged die}},\underbrace{4,3,5,2,6}_{\text{fair die}}\}.$$

Depending on whether the agent receives the roll results sequentially or can access the entire sequence directly, their prediction of what die was rolled at each time step should change. For example, since the probability distributions of the two dice overlap, the agent can only detect surprising sequences a few trials after a die change. This means that once the agent detects a change, it should retrospectively estimate the moment the task changed, before updating its models for each die.

Moreover, even if the agent observes two 2s and two 3s the first time it faces the rigged die, they cannot know whether this was an unlikely sequence with the fair die or if the die changed. However, when the agent observes the second sequence of 2s and 3s, they can create a new model for the second die, as this sequence is highly unlikely with a fair die. With this new model of the rigged die, the agent will detect future die changes more quickly. When looking back at the whole sequence with an accurate model of the rigged die, the agent may predict that the first sequence of 2s and 3s came from the rigged die, even though it did not detect the change at the time.

From this illustrative example, we extract three pivotal ideas for the model discrimination we will use. (1) An agent needs several observations to detect a task change and should not create a new model based on a single surprising observation. (2) An agent needs to be able to retrospectively detect a change point, to reassign the observations to the correct models. (3) An agent with an infinite memory could detect prior changes it missed by learning the approximate dynamics of the various models. However, storing long sequences of observations would induce memory costs. One possible solution is to work with finite-observation windows and create more models than needed in ambiguous cases, even if it means merging some models later on.

### 1.3 Contributions

We present a new multi-model method to arbitrate between local models instead of contexts. This local-model-level method allows for the transfer of knowledge between different contexts with shared sub-parts, while being more memory efficient than context-level methods. We also introduce a new technique that uses negative log-likelihood minimization to accurately detect when changes occur. We demonstrate that our multi-model agent outperforms classical reinforcement learning agents in changing uncertain environments. Our multi-model approach has limited computational cost and is stable to parameter changes.

## 2 The multi-model agent

This section focuses on describing how an agent can arbitrate between multiple local models (Figure 2). At each time step, the multi-model agent checks if its current local model adequately explains the latest local observations. If there is a stored model that better explains the last observations or if no model explains the last observations well, the agent changes its current local model. When the agent wants to change its current model, it retrospectively finds the most likely change point, finds the associated new model, and reassigns the observations after this change point to the new model. This new model becomes the current model and the agent starts using it to solve the task. At every time step, the merging module merges similar models. The forgetting module forgets a model if the agent has reached its maximal number of models and needs to create a new one.

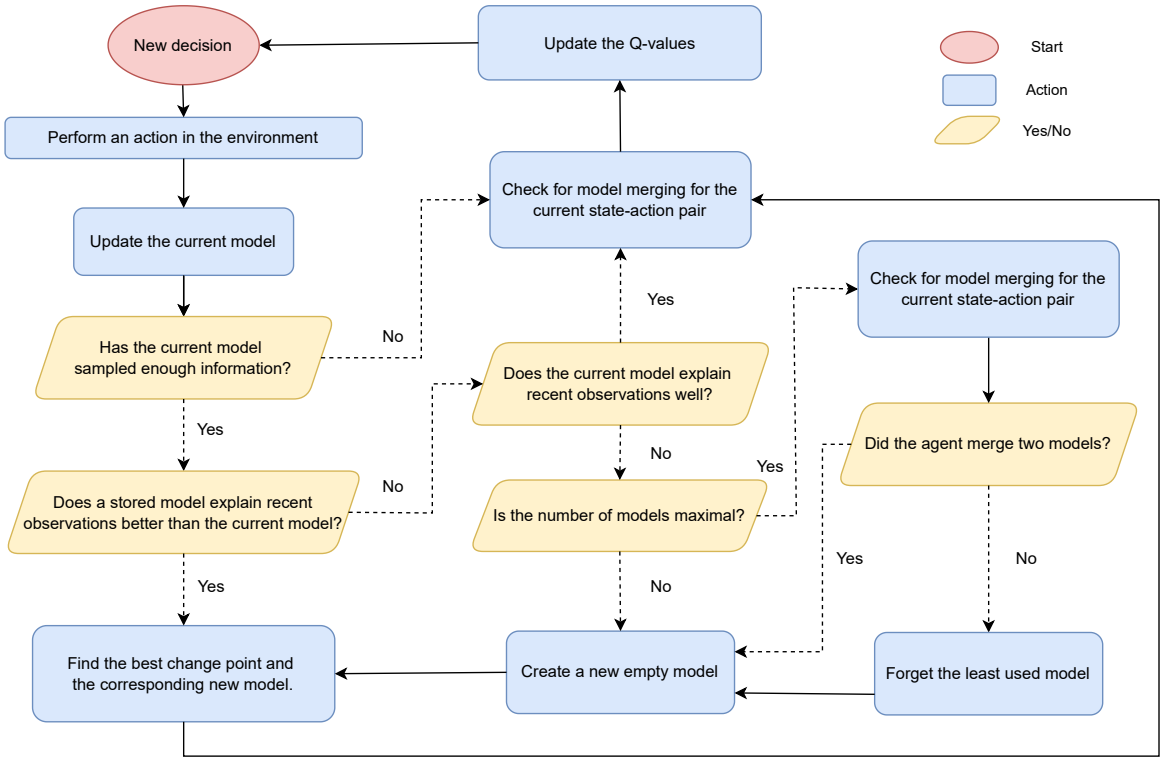


Figure 2: **Structure of the multi-model decision-making method, presented as a decision tree**. Based on yes/no tests, the tree describes what the agent does for each new decision (top-left red circle). For example, if a stored model explains recent observations better than the current model, the agent finds the best change point and changes model with the corresponding new model.

### 2.1 Updating the model

#### 2.1.1 How to learn a model in reinforcement learning?

We consider the general case of a reinforcement learning agent acting in a Markov decision process $\{S, A, T, R\}$. The agent has access to a set of states $S$ and of actions $A$. $T$ is a transition function that maps state-action pairs to the conditional probability distribution on the possible arrival states and $R$ is a reward function mapping the state-action pairs to a distribution of real values.

**Single-model infinite-horizon approach:** Classical model-based reinforcement learning agents learn an internal model of the environment for each state-action pair by sampling tuples of observations. For each state-action pair $(s, a)$, we define the reward model $M_R(s, a)$ and the transition model $M_T(s, a)$ as the set of rewards and transitions the agent has observed from performing action $a$ in state $s$. Given $(s_t,\ a_t,\ r_t)$ as the state, the action, and the immediate reward at time $t$, we denote

$$M_T(s, a) = \{s_{t+1}|s_t = s, a_t = a\}, \qquad M_R(s, a) = \{r_t|s_t = s, a_t = a\}. \tag{1}$$

An infinite-horizon model-based agent tries to estimate the true transition function $T$ and reward function $R$ based on all the observations it has sampled. The main approach to approximate $T$ and $R$ is to compute the frequency of arrival states $\hat{T}$ and the mean reward $\hat{R}$. The agent then uses $\hat{T}$ and $\hat{R}$ to plan and decide which action to take, using methods from dynamic programming such as value iteration [26]. With $n(s, a, s') = \sum_{q \in M_T(s,a)} \mathbb{1}_{q=s'}$ the number of times the agent arrived in state $s'$ after taking action $a$ in state $s$ and $n(s, a) = |M_T(s, a)| = |M_R(s, a))|$ the number of times the agent took action $a$ in state $s$,

$$\hat{T}(s, a, s') = \frac{n(s, a, s')}{n(s, a)}, \qquad \hat{R}(s, a) = \frac{1}{n(s, a)} \sum_{r \in M_R(s,a)} r. \tag{2}$$

**Single-model finite-horizon approach:** A basic approach to deal with non-stationary environments is to consider only the last $h$ observations for the transition and the reward models. The agent forgets the oldest observation of a model when the number of observations in the model exceeds $h$. We define $M_{T_h}(s, a)$ and $M_{R_h}(s, a)$ as the $h$ last rewards and arrival states the agent sampled for $(s, a)$. If the agent sampled less than $h$ observations, then $M_{T_h}(s, a)$ and $M_{R_h}(s, a)$ correspond to $M_T(s, a)$ and $M_R(s, a)$ from Equation 1. If the agent sampled $h$ observations or more, we can write $M_{T_h}(s, a)$ and $M_{R_h}(s, a)$ as

$$M_{T_h}(s, a) = \{s_{t_1}, s_{t_2}, ..., s_{t_h}|s_{t_i-1} = s, a_{t_i-1} = a\}, \quad M_{R_h}(s, a) = \{r_{t_1}, r_{t_2}, ..., r_{t_h}|s_{t_i-1} = s, a_{t_i-1} = a\}, \tag{3}$$

where $t_1 < t_2 < ... < t_h$ are the most recent $h$ time steps for which $s_{t_i-1} = s$ and $a_{t_i-1} = a$. The associated count functions are $n_h(s, a, s') = \sum_{q \in M_{T_h}(s,a)} \mathbb{1}_{q=s'}$ and $n_h(s, a) = |M_{T_h}(s, a)| = |M_{R_h}(s, a)| \leq h$. The frequentist finite-horizon transition functions $\hat{T}_h(s, a)$ and reward function $\hat{R}_h(s, a)$ become

$$\hat{T}_h(s, a, s') = \frac{n_h(s, a, s')}{n_h(s, a)} \qquad \hat{R}_h(s, a) = \frac{1}{n_h(s, a)} \sum_{r \in M_{R_h}(s,a)} r. \tag{4}$$

Finite-horizon approaches adapt better to changing or drifting environments than infinite-horizon approaches. However, the precision of their models is limited by the fixed number of passages $h$ that they consider. For example, if the true reward and transition functions are uncertain, $h$ observations may not be enough to describe them. Moreover, the model can become unstable when $h$ is low because each new observation accounts for $\frac{1}{h}$ of the estimated distribution. Finally, finite-horizon agents optimize new tasks as they come but they cannot re-use previous observations or learn multiple tasks simultaneously.

**Multi-model approach:** Another possible approach to adapt to task changes is to arbitrate between multiple competing models with infinite horizons. In this multi-model approach, the agent stores several transition and reward models and

changes its current models depending on the situation it faces. The agent associates each observed arrival state and reward to one of its transition or reward models. We define $M_T(s, a, i)$ and $M_R(s, a, i)$ as the ensemble of observations that have been attributed to the transition model $i$ and the reward model $i$ respectively. We define ${i_T}'(t)$ and ${i_R}'(t)$ as the models the agent uses for $(s, a)$ at time $t$. When taking action $a$ in state $s$, the multi-model attributes the new observations to its current reward and transition models. Hence, $M_T(s, a, i)$ and $M_R(s, a, i)$ are

$$M_T(s, a, i) = \{s_{t+1} | s_t = s,\ a_t = a,\ {i_T}'(t) = i\}, \qquad M_R(s, a, i) = \{r_t | s_t = s,\ a_t = a,\ {i_R}'(t) = i\}. \tag{5}$$

Each model has an infinite memory and the agent can use Equation 2 to compute an approximate transition function $\hat{T}$ and reward function $\hat{R}$.

#### 2.1.2 The update module

The agent we introduce uses a multi-model approach. At each time step, the agent adds the observed reward and arrival state to its current reward model and transition model, while the other reward and transition models do not change as presented in Equation 5. The multi-model agent also maintains a record of the most recent arrival states and rewards received from state $s$ by taking action $a$ in the last $h$ steps. It also keeps track of the local model it was using at the time of each observation. Depending on how closely the agent's transition model or reward model aligns with the observed data of the last $h$ steps for $(s, a)$, the agent will either continue updating the current local model, change to a stored model or create a new one.

We present the methods and results of the multi-model agent when it arbitrates between transition models. However, these same methods can also be applied to reward models, provided the agent uses a distributional reward function. We chose to focus on transition distributions because the classical transition function we introduce in Equation 2 directly provides a distribution (a histogram) over the arrival states.

### 2.2 Detecting a change

#### 2.2.1 How to evaluate model reliability with incoming observations?

The goal of the task change detection method is to find when the current model can no longer explain recent observations. The classical approach to check if a distribution explains incoming data is to use statistical tests or information theory methods. Here, we aim to test whether a model can accurately explain $h$ observations, where $h$ is small (*e.g.*, about 10 observations). Since this sample size may be too small to run valid statistical tests, we instead rely on information theory methods.

The main information theory approach to assess model reliability in similar work is to use log-likelihood [19, 5]. The agent evaluates the likelihood of each new observation based on the predictions of its current model. However, using log-likelihood has shortcomings. First, the computation of the log-likelihood depends on the true dynamics of the environment. If the environment's dynamics are uncertain, predicting the observations becomes less likely, even if the models are accurate. One potential solution is to use the difference in log-likelihoods to evaluate how much the agent progresses in explaining recent observations. However, log-likelihoods may not always detect changes in the incoming data. For example, if the model is a fair die, the likelihood of observing several sixes in a row is the same as observing any other sequence of the same length. To address this limitation, the multi-model agent does not use log-likelihoods to measure model reliability. Instead, it directly compares distributions.

The agent wants to compare the stored distributions with the distribution of the recent observations. There is an asymmetry between these two distribution types, as the stored models are generally larger and more stable than recent observations. A symmetric measure such as the Wassertein distance or the Jensen-Shannon divergence would not account for this discrepancy. Conversely, the Kullback-Leibler divergence measures how well a well-established model explains recent observations. With two probability distributions $P$ and $Q$, of support $X_P$ and $X_Q$, with $X_P \subset X_Q$, the Kullback-Leibler divergence $D_{KL}(P, Q)$ is

$$D_{KL}(P, Q) = \sum_{x \in X_P} P(x) \log \frac{P(x)}{Q(x)}. \tag{6}$$

In our scenario, $P$ is the distribution from the recent observations and $Q$ the distribution from a model learned by the agent.

### 2.2.2 The change detection module

The change detection module checks at each time step whether the current model still provides the best explanation for the most recent observations. If a stored model explains the recent observations better than the current model, or if the current model fails to explain the current observations, the agent must change its current model.

We suppose that for a given state and action $(s, a)$, the agent stored $k$ models $M_{T_{1 \leq i \leq k}}(s, a)$ and that the current model is model $i'$. Every time the agent takes action $a$ in $s$, it computes the Kullback-Leibler divergence between each stored model and the distribution of the last $h_{i'}$ observations, with $h_{i'}$ being the last consecutive observations attributed to the current model $i'$ in the last $h$ passages. Thus $h_{i'} \leq h$ and $h_{i'} = h$ if the last $h$ passages were attributed to model $i'$. The agent computes the Kullback-Leibler divergence from Equation 6 between the distribution of the last observations $\hat{T}_{h_{i'}}(s, a)$ and the distribution of each stored model $\hat{T}_i(s, a)$.

$$D_{KL}(\hat{T}_{h_{i'}}(s,a), \hat{T}_i(s,a)) = \sum_{s'/\hat{T}_{h_{i'}}(s,a,s')>0} \hat{T}_{h_{i'}}(s,a,s') \log \frac{\hat{T}_{h_{i'}}(s,a,s')}{\hat{T}_i(s,a,s')}. \tag{7}$$

In Equation 7, if the support of $\hat{T}_{h_{i'}}(s, a)$ is not included in $T_i(s, a)$, the KL divergence diverges to infinity. In practice, we add a small value $\delta = 10^{-5}$ to each distribution for all the state-action pairs of the environment the agent knows, and normalize each distribution so that the KL divergence is well defined. The KL divergence of the current model is always well-defined because the support of model $i'$ contains the latest $h_{i'}$ observations.

If a known model explains the last observations better than the current model, the agent should replace its current model with a stored one. If the current model does not explain the recent observations well, but is better than all stored models, the agent should create a new model. Finally, if the current model has the minimal KL divergence and the divergence is low enough, the agent should keep using its current model. We sum up these three cases in Equation 8. With $1 \leq i' \leq k$ the current model, and the model creation threshold $\Delta_c$,

$$\begin{cases} \text{if } D_{KL}(\hat{T}_{h_{i'}}(s,a), \hat{T}_{i'}(s,a)) \neq \min\limits_{1 \leq i \leq k} D_{KL}(\hat{T}_{h_{i'}}(s,a), \hat{T}_i(s,a)), \text{ change the model;} \\ \text{else if } D_{KL}(\hat{T}_{h_{i'}}(s,a), \hat{T}'_i(s,a)) > \Delta_c, \text{ create a new model;} \\ \text{else, stick with the current model.} \end{cases} \tag{8}$$

This method can detect a change but does not indicate when the change occurred. If the agent decides to swap its current model with a stored one, using directly the one with a minimal Kullback-Leibler divergence can lead to sub-optimal behavior. We explain how the agent can overcome this limitation in the next subsection which focuses on change point detection.

## 2.3 retrospectively finding the change point

### 2.3.1 Why should the agent find a change point?

The previous subsection presented how to detect a change and find the model that best explains the most recent observations. However, choosing the new model or context based on all recent observations tends to favor moderate models.

To illustrate this phenomenon, we take the example of a passive observer modeling a coin that can land on heads ($H$) or tails ($T$). The observer sees the sequence $\{H, H, H, H, T, T, T\}$, with tails being the latest observation. The agent has three coin models: model 1, which is its current model and mostly predicts heads, model 2, which mostly predicts tails, and model 3, which predicts heads or tails with the same probability. After the last observation, model 3 becomes the model which explains best the observed sequence. Thus, the agent changes its current model. However, if the agent changes its current model based on all recent observations, it should use model 3, although it poorly explains the most recent three observations. Instead, the agent must detect that the swap occurred between Model 1 and Model 2 after the last heads and before the first tails. retrospectively finding this change point allows the agent to choose the appropriate associated model for the task.

A second implication of precise change point detection is that the agent can reassign observations to the correct models depending on when the change happened. Observations occurring after the change point belong to the new model and should be reassigned from the current model to the new model. This improves both models: the former becomes more precise by discarding incorrectly assigned observations, while the latter benefits from additional relevant experiences.

### 2.3.2 The change point detection module

To find the change point between the current model and a new one, the agent assesses to what extent it can sequentially redirect the latest observations from its current model to the new model (Figure 3). It does this by gradually computing the log-likelihood of recent observations for both the current model and each potential new model. One can implement other approaches to evaluate the likelihood of the change points, such as a leave one out log-likelihood based on change point's position. However, the agent must evaluate all possible change points and new models, making more computationally intensive methods less practical than the approach we propose.

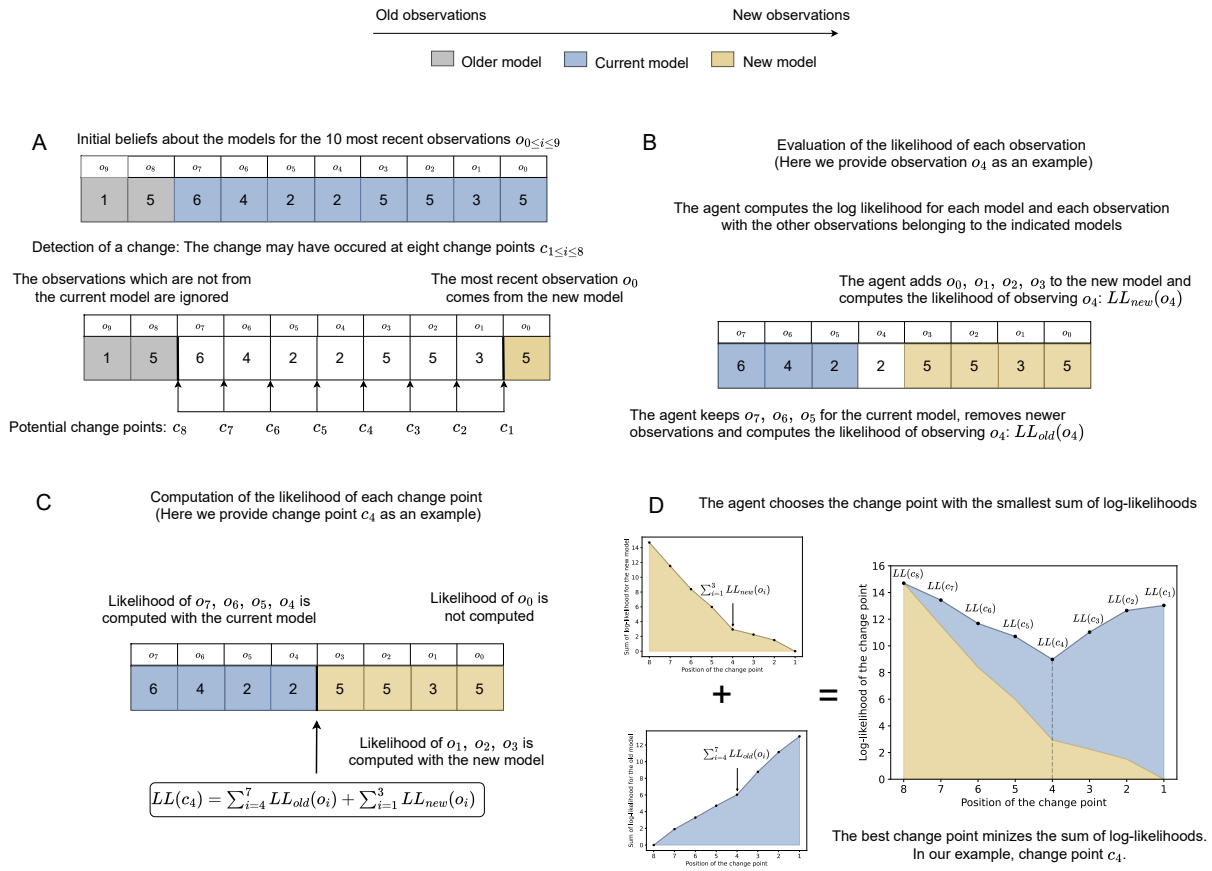


Figure 3: **Illustration of the method to find a change point.** The two distributions we compare are the top ones from Figure 4. When detecting a change, the agent follows these four steps (A to D) in order. **Panel A** The horizon has a size of $h = 10$, but the horizon for the current model alone is $h_{i'} = 8$ because an older model is used for the two first observations. As a result, the change point can only be at one of 8 possible positions. **Panel B** The agent computes the likelihood of each observation, for the two models. **Panel C** The total evaluation of each change point is the sum of the individual log-likelihoods the agent computed individually. **Panel D** The agent compares the log-likelihood of all change-points. The change point which minimizes the sum of log-likelihoods in this example is change point 4.

We denote $\{s'_{h_{i'}}, ...s'_2, s'_1\}$ as the most recent arrival states the agent has observed from performing action $a$ in $s$. Here, $s'_1$ is the most recent observation while $s'_{h_{i'}}$ is the oldest. There are $h_{i'} - 1$ possible change points between two successive observations and two possible change points before or after all the observations. Nevertheless, the last observation belongs to the new model. Hence, there are $h_{i'}$ possible change points, going from only the last observation belonging to the new model to all of the last $h_{i'}$ observations belonging to the new model.

To compute the log-likelihoods, the agent needs to be able to add the observations to the new model and to remove them from its current model. When the agent adds a new observation $s'_j$ to a model $i$, the agent updates the transition function using Equation 9.

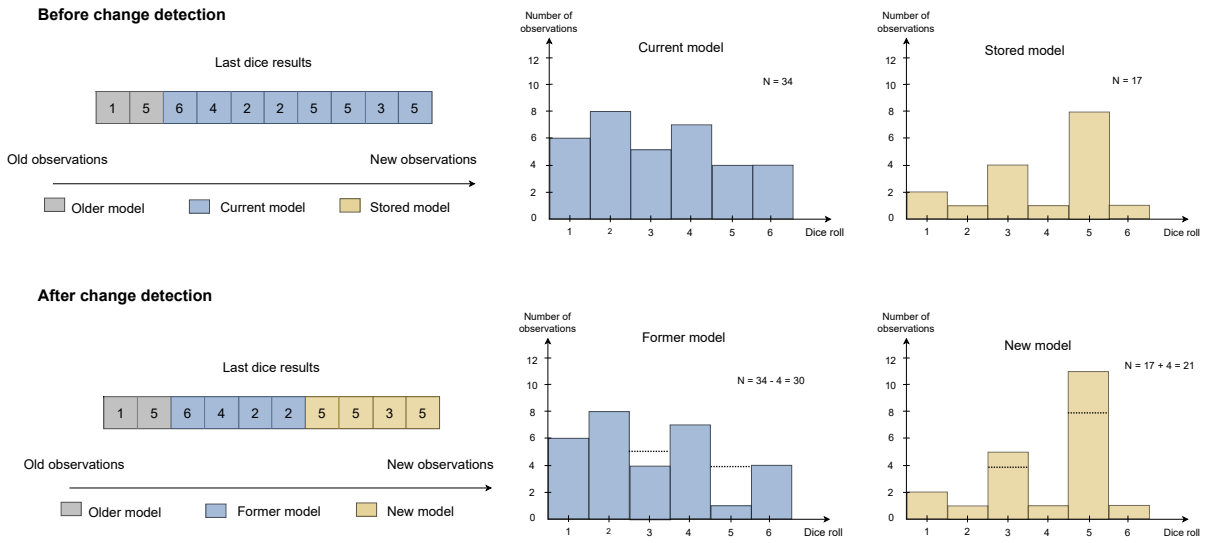


Figure 4: **Illustration of the reassignment of the observations after finding a change point**. (Left) The change point and the last observations come from Figure 3. (Top center and right) Histograms of the die rolls before the model change detection for the current model and a stored model. We make the assumption that the agent detects a change from the current model to the stored model. (Bottom center and right) Histograms after the model change detection. The current model becomes the former model and the stored model becomes the new model. The histograms illustrate the reassignment of the observations from the former model to the new model.

$$n_i^{+s'_j}(s,a) = n_i(s,a) + 1, \qquad T_i^{+s'_j} = \frac{n_i^{+s'_j}(s,a,s'_j)}{n_i^{+s'_j}(s,a)}. \tag{9}$$
$$n_i^{+s'_j}(s,a,s'_j) = n_i(s,a,s'_j) + 1,$$

To remove an observation $s'_j$ from a model $i$, all the pluses from Equation 9 become minuses. We extend Equation 9 to any ensemble of arrival states $A(s,a)$. We denote $T_i^{+A(s,a)}$ as the transition function of model $i$ with the arrival states $A(s,a)$ added to model $i$. Similarly, we denote $T_i^{-A(s,a)}$ as the transition function when the arrival states $A(s,a)$ are taken away from model $i$. Removing an observation from a model can only happen if the model has this observation in its model.

To compute how well the current model $i'$ explains the recent observations, the agent computes the log-likelihood of witnessing each observation supposing that the model used before the observation was $i'$. Starting with the second most recent arrival state $s'_2$, the agent computes its new transitions without $s'_1$ and $s'_2$ and computes the log-likelihood of observing $s'_2$ with model $i'$. Similarly, the agent computes the likelihood of observing each arrival state $s'_j$ with $2 \geq j \geq h_{i'}$ after removing $s'_j$ from its model and all the following observations. With $S'_{1\to j} = \{s'_j, ..., ..., s'_2, s'_1\}$, the ensemble of the last $j$ arrival states, the likelihood of observing $s'_j$ with the current model $i'$ is

$$LL(T_{i'}^{-S'_{1\to j}}, s'_j) = -\log T_{i'}^{-S'_{1\to j}}(s,a,s'_j). \tag{10}$$

With this method, the agent can retrospectively compute the likelihood of observing each of the last $h_{i'}$ events with the model it had at the time of the observation. The agent does a similar computation for all the possible new models by adding the observations instead of taking them away from the model. Starting with the most recent observation $s'_1$, the agent computes the probability of observing $s'_2$. Similar to Equation 10, the likelihood of observing each event $s'_j$ with the model $i \neq i'$ is

$$LL(T_i^{S'_{1\to j}}, s'_{j+1}) = -\log T_i^{S'_{1\to j}}(s,a,s'_{j+1}). \tag{11}$$

In Equation 10 and Equation 11, the agent adds $\delta = 10^{-5}$ to each distribution and normalizes them before computing the log-likelihood. This ensures that all the log-likelihoods are well defined. From Equation 10 and Equation 11, the

agent can compute the likelihood of each change point. The likelihood of a change point $c$ between model $i'$ and model $i$ is given by the likelihood that observations before $c$ belong to the current model $i'$, while those after $c$ belong to the new model $i$. For each couple of models $(i', i)$, the likelihood $L_i(c)$ of change point $c$ with $1 \leq c \leq h_{i'}$ reads

$$L_i(c) = \sum_{j=1}^{c-1} LL(T_i^{S'_{1\to j}}, s'_{j+1}) + \sum_{j=c}^{h_{i'}-1} LL(T_{i'}^{-S'_{1\to j+1}}, s'_{j+1}). \tag{12}$$

The optimal change point $c_{opt}$ is defined as

$$c_{opt} = \underset{c}{\operatorname{argmin}} \min_i L_i(c). \tag{13}$$

This method provides the new model $i_{opt}$ and the optimal change point $c_{opt}$ (Figure 3). Once the agent finds the best change point and the best corresponding model, it reassigns the observations after the change point from its current model to the new model and starts using it (Figure 4).

### 2.4 Merging models

#### 2.4.1 How to know when to merge two distributions together?

There are several ways to compare two distributions. However, the choice of metrics depends on the goal of the comparison and the nature of each distribution. In this case, an asymmetric measure is unnecessary since both models represent stable beliefs about the environment. The goal of the comparison is to merge the two models into a unified model that encompasses both. Thus, instead of directly comparing the models, we can measure how far each model is from a combined distribution.

A common symmetric distance to compare distributions to their average distribution is the Jensen-Shannon divergence. The Jensen-Shannon divergence computes the Kullback-Leibler divergences of the two distributions to their average distribution and sums them with an equal factor of $\frac{1}{2}$. The Jensen-Shannon divergence of two probability distributions $P$ and $Q$ is defined as

$$D_{JS}(P, Q) = \frac{1}{2} D_{KL}(P, \frac{P+Q}{2}) + \frac{1}{2} D_{KL}(Q, \frac{P+Q}{2}). \tag{14}$$

However, the Jensen-Shannon divergence in Equation 14 takes the average of the two probability distributions as a merged distribution. In our scenario, we want the merged model to account for the relative sizes of the two distributions. For example, a model with more observations should have a greater weight in the merging process. Instead of comparing the two distributions to their average, we compare them to a weighted merged distribution where each distribution's influence is proportional to its number of observations. We show an example of a merged model in Figure 5.

A distance measure inspired by the Jensen-Shannon divergence, which weighs the initial distributions during merging is the weighted Jensen-Shannon divergence. This method provides theoretical guarantees, such as an upper bound on the divergence [9]. It also accounts for the size of the distributions in the divergence computation by replacing the $\frac{1}{2}$ factors from Equation 14 with weighted factors that depend on the sizes of the two distributions. Given $N_P$ and $N_Q$ as the weights of distributions $P$ and $Q$ (*e.g.*, their respective model sizes), the weighted Jensen-Shannon divergence is defined as

$$D_{wJS}(P, Q) = \frac{N_P}{N_P + N_Q} D_{KL}(P, \frac{N_P P + N_Q Q}{N_P + N_Q}) + \frac{N_Q}{N_P + N_Q} D_{KL}(Q, \frac{N_P P + N_Q Q}{N_P + N_Q}). \tag{15}$$

However, when one distribution has significantly more observations than another one, their weighted Jensen-Shannon divergence is close to 0 (Equation 15). In our scenario of model merging, weighting the Kullback-Leibler divergences by the number of observations would lead to large models overpowering smaller ones, reducing model diversity. To address this, a more suitable approach is the semi-weighted Jensen-Shannon divergence, which uses weighted merging but keeps the Kullback-Leibler divergence factors unweighted. The weighted merging Jensen-Shannon divergence is

$$D_{swJS}(P, Q) = \frac{1}{2} D_{KL}(P, \frac{N_P P + N_Q Q}{N_P + N_Q}) + \frac{1}{2} D_{KL}(Q, \frac{N_P P + N_Q Q}{N_P + N_Q}). \tag{16}$$

Using the semi-weighted merging Jensen-Shannon divergence from Equation 16 has several many implications. First, we lose some theoretical properties of the Jensen-Shannon divergence and its weighted alternative, such as upper bounds. However, this method preserves model diversity while incorporating model sizes into the merging process. If one model $P$ is much bigger than one model $Q$, the semi-weighted Jensen-Shannon divergence is close to a Kullback-Leibler divergence (with a factor $\frac{1}{2}$). Unlike the Kullback-Leibler divergence, the semi-weighted Jensen-Shannon divergence is always well-defined here because the merged distribution includes the observations from both distributions. If the two distributions are of the same size, the semi-weighted Jensen-Shannon divergence reduces to the Jensen-Shannon divergence.

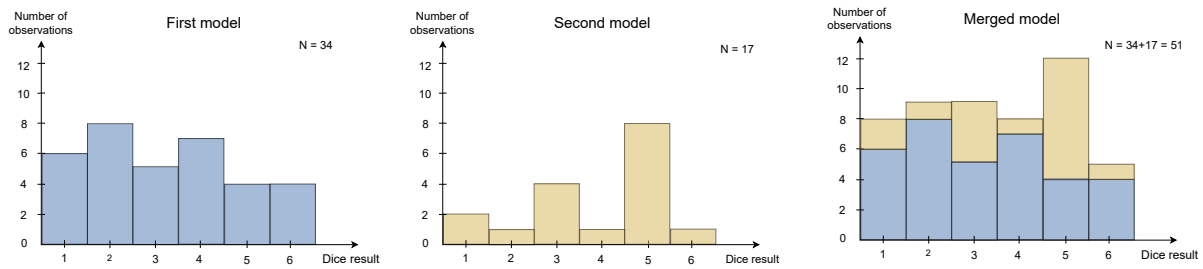


Figure 5: **Example of the model merging process.** If the initial models are similar to the merged model, the agent merges the two models by creating a model that includes all the observations. One can compute the probability distribution for each plot by normalizing the data by the the number of observations $N$.

### 2.4.2 The merging module

After every action $a$ in state $s$, the agent checks if two models of $(s, a)$ can be merged. If so, the agent merges the two models and continues merging models until no model can be merged anymore. Merging models limits the computational and memory cost of the agent by reducing the numbers of models to compare.

When the agent checks whether it can merge two models or not, it computes a semi-weighted Jensen-Shannon divergence between each couple of probability distribution. If the smallest semi-weighted Jensen-Shannon divergence is below a merging threshold $\Delta_m$, the agent merges the two corresponding distributions. To compute the weighted Jensen-Shannon divergence between two transition distributions $T_i(s, a)$ and $T_j(s, a)$, the agent first sums the observations of the two models together $n_{i+j}(s, a, s') = n_i(s, a, s') + n_j(s, a, s')$ for all the arrival states $s'$ (Figure 5). From these counts, the agent computes the corresponding distribution $T_{i+j}(s, a) = \frac{n_{i+j}(s,a,s')}{n_{i+j}(s,a)}$. For transition models, Equation 16 becomes

$$D_{swJS}(T_i(s, a), T_j(s, a)) = \frac{1}{2} D_{KL}(T_i(s, a), T_{i+j}(s, a)) + \frac{1}{2} D_{KL}(T_j(s, a), T_{i+j}(s, a)). \tag{17}$$

When the agent merges two models, it replaces one model with the merged model and forgets the other one, reducing the total number of models by one. The agent also reallocates all observations from the discarded model to the merged model.

## 2.5 Forgetting a model

The ability to forget a model allows the agent to set a maximum number of models $\max_{mod}$. Limiting the number of models reduces both memory and computational costs, as the number of comparisons increases with the number of models. This limit can also align with biological constraints. For example, some participants in the working memory task where the PROBE model was first tested only maintained and evaluated three models simultaneously [8].

In the multi-model agent, forgetting a model occurs only when the agent reaches $\max_{mod}$ and cannot merge any models. In this case, it discards the model with the fewest observations. If the current model is the smallest, the agent instead forgets the second smallest model. Retaining the current model ensures stability and reflects the idea that recent observations are more salient than older ones.

We chose to implement this forgetting mechanism for its simplicity. However, since the agent cannot forget its current model, it must be able to retain two local models per state-action pair and distribution type (transition or reward). In our simulations, the threshold $\max_{mod}$ depends on the state-action pair and the agent can store the same number of models per state-action pair. This forgetting module implies that $\max_{mod} \geq 2$.

There are at least two alternative implementations for the forgetting module. First, the threshold $\max_{mod}$ could become a global variable rather than state-action specific. This would allow the agent to store more models in areas with frequent changes. Second, instead of forgetting the smallest model, the agent could forget the least recently used one, as unused models may be irrelevant. The choice of implementation may depend on the task at hand.

# 3 Experiments

## 3.1 Comparison of the multi-model agent with other reinforcement learning agents

The multi-model agent can use any decision-making or exploration strategy, as arbitration between models is independent of these processes. Nevertheless, to simplify the comparison, all simulated agents use a similar soft-max decision-making process. The probability of choosing action $a$ in state $s$ is

$$P(a|s) = \frac{\exp^{\beta Q(s,a)}}{\sum_{a'} \exp^{\beta Q(s,a')}}, \tag{18}$$

where $\beta > 0$ is an inverse temperature factor and $Q(s, a)$ is a Q-value. The agent updates the Q-values with the Bellman equation based on the model it learns online, as follows

$$Q(s, a) = \hat{R}(s, a) + \gamma \sum_{a'} \hat{T}(s, a, s') \max_{a'} Q(s', a'), \tag{19}$$

with $0 < \gamma < 1$ a discounting factor. We use $\gamma = 0.95$ for all the agents.

We highlight the advantages of the multi-model agent by comparing it with two other reinforcement learning agents, each using a different type of model. We compare the multi-model agent to an infinite-horizon agent with a single model per state-action pair (Equations 1 and 2) and to a finite-horizon agent with a single model per state-action pair (Equations 3 and 4). While the multi-model agent uses infinite-horizon models, it uses a finite horizon to arbitrate between models. The goal of this simple comparison is to validate the general behavior and stability of the multi-model agent.

## 3.2 Nature of the task changes

In the reinforcement learning literature, changes can occur at different levels and times [17]. The multi-model agent can theoretically adapt to changes at any level, from local variations to broader context shifts, provided they do not happen too frequently. For example, if one local change happens faster than the study horizon $h$, the agent cannot detect this local change and averages out changes. To simplify our analysis of the multi-model agent, we use environments where dynamics change at a fixed rate. However, the multi-model agent neither knows nor relies on this information. The number of states and actions in a chosen environment remains constant over task changes.

All of the changes we present in this article pertain specifically to transitions. Hence, the multi-model agent maintains a reward model for each transition model and does not try to detect changes in reward. When the agent changes its transition model, it changes its reward model accordingly and reassigns the rewards depending on the change point.

## 3.3 Environments

To measure and compare the performance of the three agents, we use three types of environments of increasing difficulty. The first environment is a small environment with three states and two actions (Figure 6, top). The second one is an adapted version of the chain problem [11, 24] for non-stationary tasks (Figure 6, bottom). The third one consists of an ensemble of ten maze environments with random wall generation (Figure 7).

**The three-state environment -** In the three-state environment (Figure 6, top-left), the goal of the agent is to reach the state with a positive reward $r = 1$. The *Left* and *Right* actions are swapped at a fixed rate. The agent has a probability of slipping $p_{slip} = 0.1$, which causes it to experience the outcome (reward and arrival state) of the opposite action. After every step, the agent is reset to the initial state.

**The chain environment -** In the chain environment, the goal of the agent is to reach the farthest state from its starting state to get a maximal reward of $r = 1$. Every-time the agent chooses the action *Left*, it goes back to the initial state where it receives a bait reward of $0.1$. If the agent reaches the farthest state from its starting position, it receives a reward of $r = 1$. The optimal policy is to do the action *Right* only. At each step, the agent has a probability of slipping

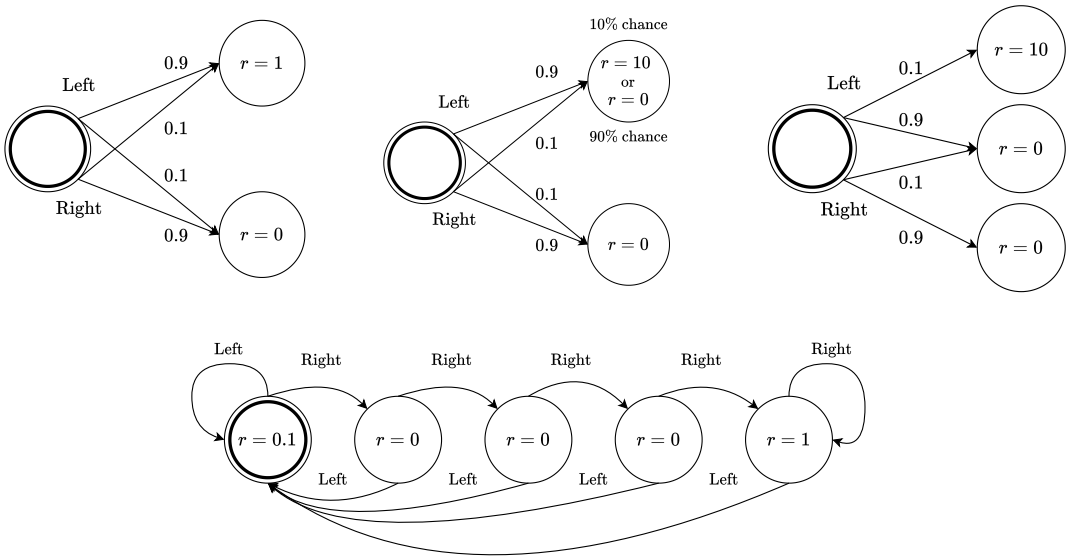


Figure 6: **The three-state environment, two variations of it, with reward uncertainty or transition uncertainty, and the chain task.** Each of the two actions has the probability $p_{slip} = 0.1$ of resulting in the opposite action. The black circles indicate the starting position of the agent. (Top-Left) The three-state environment. (Top-Center) The three-state environment with an uncertain reward distribution on the rewarded state. (Top-Right) The three-state environment with an additional fourth state, to add an infrequent yet key transition. (Bottom) The chain environment. The slipping probabilities $p_{slip} = 0.1$ are not represented for readability.

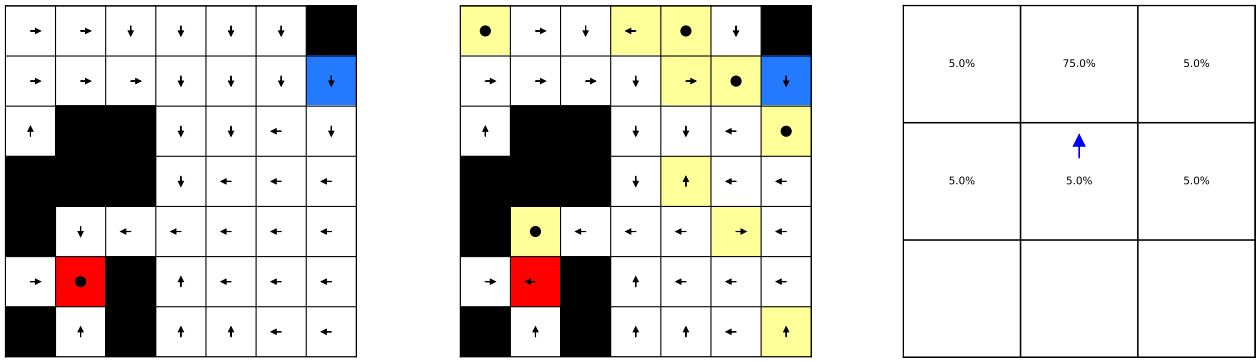


Figure 7: **One of the mazes the agent faces.** (Left) One of the 10 mazes the agent faces of size $7 \times 7$, with randomly generated walls. The initial state is in blue, the rewarded state is in red and the walls are in black. The one-step movements are indicated with arrows and the *Stay* action with a filled circle. (Middle) For each task change, each cell has a 20% chance of changing its action distributions, performing a predefined cyclic permutation over the actions. The states whose actions have been permuted are highlighted in yellow. (Right) Representation of the arrival state distribution when performing action *Up* in absence of walls. The action *Stay* is deterministic.

$p_{slip} = 0.1$ and receiving the outcome (arrival state and thus reward) of the opposite action. The chain environment we implemented has 5 states and 2 actions. The task-change we introduce is a local task change, with the *Right* and *Left* action inverted for one state. Thus, there are $2^5 = 32$ possible combinations or contexts.

**The maze environments -** The maze environments are $7 \times 7$ squared environments with random wall generation (Figure 7). In the maze environments, the goal of the agent is to reach the maximal reward of 1 and to exploit it. The cell with the reward is 10 cells away from the initial cell. The agent can perform 4 one-step movement actions (*Up*, *Down*, *Left*, *Right*) and one *Stay* action. The transitions are stochastic: when the agent performs one of the 4 one-step movement actions, it can end up in one of 6 possible cells. It has a 75% chance of reaching the state corresponding to the deterministic action and a 5% chance of ending in the nearby cells as presented in Figure 7. If there is a wall in the

destination cell, the agent stays where it is. The stay action is deterministic with a $100\%$ chance of staying in the same cell. For each task change, there is a $20\%$ chance of performing a predefined cyclic permutation of the actions for each cell. There are two possible distributions per cell (either the classical one or a cyclic permutation of it). Thus, there are theoretically $2^{|S_{nw}|}$ possible contexts, with $|S_{nw}|$ the non-wall states of the maze. The environment in Figure 7 has $2^{39}$ possible contexts. Therefore, a context-level agent would not be able to learn the optimal policy in all the different contexts.

**Variations of the three-state environment and the mazes -** To show the interest of learning accurate models in changing environments, we add uncertain transitions or rewards to the three-state environment and the mazes. In the three-state environment, we add an uncertain reward or an uncertain transition. The uncertain reward consists in rewarding the agent $10\%$ of the time with $r = 10$, instead of $r = 1$ every time the agent reaches the appropriate state (Figure 6, top-center). The uncertain transition consists in adding a fourth state the agent reaches $10\%$ of the time when choosing the right action and where it gets a reward of $r = 10$ (Figure 6, top-right). In both conditions, the agent must learn a rare event to find the optimal policy. For the mazes, we add an uncertain transition event the agent must learn to ignore. For all the actions, the agent has a $5\%$ chance of being reset back to its starting position. The goal of these uncertain transitions or rewards is to identify which agents learn accurate transition and reward distributions, while adapting to changes.

# 4 Results

All simulations from the results section show the mean performance and the confidence interval at $95\%$ of the multi-model agent, the finite-horizon agent, or the infinite-horizon agent. We evaluate the agents over three metrics: the reward accumulated, the computational time and the Euclidean distance to the true transitions of the environment. We plot the performance of the agents over time and averaged over each trial following a task change, to inspect how the agent relearns after a change. When one of these plots is not presented in the main article, it is available in the supplementary information section A.

## 4.1 Comparison of the multi-model agent with the other agents

### 4.1.1 Performance in the three-state environment

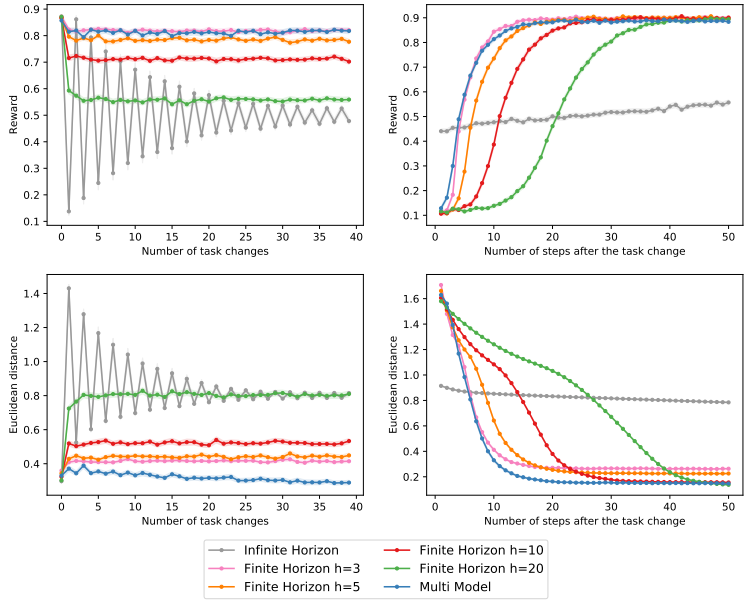


Figure 8: **Comparison of the performance of the agents in the three-state environment.** Each agent is evaluated over 200 seeds. (Top-Left) Rewards obtained in the three-state task, averaged over each task change. (Top-Right) Rewards obtained in the three-state task, averaged over each of the 50 step between task changes. (Bottom-Left) Euclidean distance between the learned model and the real transitions, averaged over each task change. (Bottom-Left) Euclidean distance between the learned model and the real transitions, averaged over each of the 50 step between task changes.

To demonstrate the interest of the multi-model agent in a simple environment, we run simulations in the three-state environment. We evaluate each agent on 200 seeds, with an episodic setting of 1 step and 2000 trials: after each step, the agent is reset to the starting state. We swap actions *Left* and *Right* every 50 steps. All agents have the same exploration rate $\beta = 10$ which corresponds to a greedy behavior. The finite-horizon agents use an horizon of $h = 3$, $h = 5$, $h = 10$, or $h = 20$. The multi-model agent uses $h = 5$, $\Delta_c = 0.3$, $\Delta_m = 0.3$, and $\max_{mod} = 5$, which corresponds to high model creation and merging rates.

The multi-model and the finite-horizon agents adapt to the task changes (Figure 8). The multi-model performs as good as the finite-horizon agent with an horizon $h = 3$, as they detect the task change the fastest. The finite-horizon agents with longer horizons take longer to adapt to task changes. Since $\beta$ is high, the finite-horizon agents exploit their model of the last transitions for several steps before exploring the other option. The infinite-horizon agent ends up averaging all the changes, leading to a close to random behavior.

The finite-horizon agent with $h = 3$ performs as well as the multi-model agent. However, this agent does not learn accurate transition dynamics, as opposed to the multi-model agent (Figure 8). In this simple task, learning accurate transition dynamics is not mandatory to find the optimal policy.

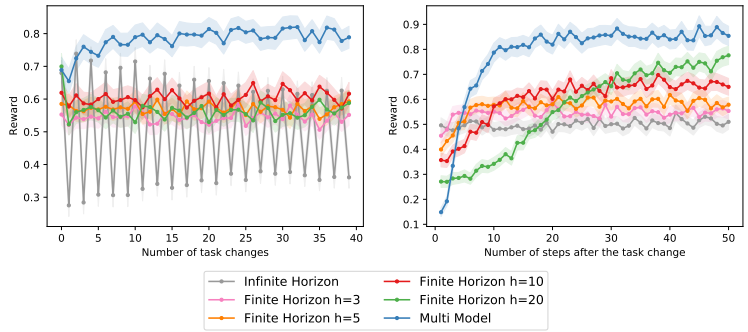


Figure 9: **Comparison of the performance of the three agents in the three-state environment with uncertain rewards.** Each agent is evaluated over 500 seeds. (Left) Rewards obtained averaged over each task change. (Right) Rewards obtained averaged over each of the 50 steps between task changes.

With an accurate model of the environment, the multi-model agent can track uncommon yet key events, such as rare transitions or rare rewards. We reproduce the experiment presented in Figure 8 with uncertain reward. Instead of receiving a reward of 1 with a 100% chance, the agent receives reward of 10 with a 10% (Figure 6, top-center). We evaluate each agent over 500 seeds.

The multi-model agent learns the task over time and adapts well to task changes, contrary to finite-horizon and infinite-horizon agents (Figure 8). With a small horizon, the finite-horizon agent does not learn an accurate reward function. With a large horizon, it does not detect task changes fast enough. Conversely, the multi-model agent keeps infinite and accurate models of the rewards and solves the task. Agents with a finite horizon in transitions but an infinite horizon in rewards may find the optimal policy, provided they learn state-dependent reward functions. Therefore, we also demonstrate this phenomenon with rare but key transitions.

To demonstrate the interest of the multi-model approach for transition uncertainty, we add a fourth state to the environment. To get a deterministic reward of $r = 10$, the agent must find this fourth state. However, only one of the two actions leads to this state, with a 10% chance (Figure 6, top-right). Thus, the agent needs to learn an accurate model of the transitions to find the optimal policy. Only the multi-model agent finds the optimal policy (Figure 10). The finite-horizon agents do not learn accurate models of the environment and have to re-learn the 10% chance of getting a reward every time the task changes. In contrast, the multi-model agent loads an accurate model of the transitions when detecting a task change.

The multi-model agent performs well in situations where infinite-horizon and finite-horizon single-model agents struggle (Figures 9 and 10). Moreover, the multi-model agent uses the same set of parameters for the three simulations. Conversely, the best horizon length for the finite-horizon changed based on the task's uncertainty.

#### 4.1.2 Performance on the chain task

The second test environment is the chain task, with $p_{slip} = 0.1$ and a 5 states chain. The *Left* and *Right* actions are swapped in one random state every 10 trials of 50 steps and over 50 task changes. All agents use $\beta = 3$ and are

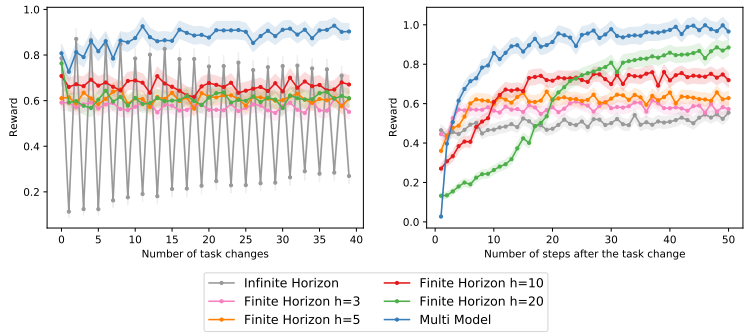


Figure 10: **Comparison of the performance of the three agents in the three-state environment with a rare transition.** Each agent is evaluated over 500 seeds. (Left) Rewards obtained averaged over each task change. (Right) Rewards obtained averaged over each of the 50 steps between task changes.

evaluated over 100 seeds. We set $h = 3$, $h = 5$, $h = 10$, or $h = 20$ for the finite-horizon agents. The multi-model agent uses $h = 10$, $\Delta_c = 1$, $\Delta_m = 0.1$, $\max_{mod} = 5$.

Similar to the first task, the infinite-horizon agent averages out the changes. Thus, its reward and computational time decrease while its distance to the true transitions increases after a few task changes. The finite-horizon and the multi-model agents solve the task with similar performances. Shorter finite-horizon agents find a good yet non-optimal policy faster than longer finite-horizon agents. The computational costs of the multi-model and the finite-horizon agents are similar. The multi-model agent learns slightly better the true transition dynamics than the finite-horizon agent. This effect occurs over time, as the multi-model agent learns the transitions accurately, and after a task change, as the multi-model agent loads the true dynamics of the environment after detecting a task change. As in Figure 8, knowing the transitions perfectly is not mandatory to find the optimal policy. However, short-term horizon which do not learn an accurate model of transitions fail to adopt the optimal behavior.

#### 4.1.3 Performance in the mazes

The mazes are large environments to assess the scalability of the multi-model approach. For each trial, all the agents have 100 steps to find and exploit the reward. They are evaluated over 1000 trials, with a task change every 20 trials (or 2000 steps). The transitions from a state can be in two modes: either the classical action distributions or a cyclic permutation of it. A task change consists in selecting approximately 20% of the states and changing the current mode to the other one. All agents use $\beta = 3$ and the multi-model agent uses the same parameters as in the chain task: $h = 10$, $\Delta_c = 1$, $\Delta_m = 0.1$, and $\max_{mod} = 5$. We test finite-horizon agents with $h = 5$, $h = 10$, $h = 20$, and $h = 30$.

The multi-model agent performs as well as the finite-horizon methods in the mazes (Figure 12). However, the multi-model is more computationally costly. Understanding accurate dynamics of the task is not mandatory to find the optimal policy. Thus, finite-horizon with a short-horizon perform better than finite-horizon agents with longer horizons which are slower at detecting the task change. The infinite-horizon agents fail to adapt to task changes.

To demonstrate the interest of learning accurate dynamics of the transition distributions, we add noise in the transition dynamics of the mazes. For all the actions, the agent has a 5% chance of going back to the initial state. The agent must now understand that this rare transition should be ignored to find the optimal policy. Adding uncertainty on the transition distributions is similar to the change introduced for the three-state environment (Figure 10). However, the uncertain transition is on all the state-action pairs in the mazes and is detrimental to the agent's performance. Conversely, the uncertain transition was rewarding in the three-state environment.

Adding a punishing and rare transition decreases the performance of all the agents (Figure 13). The performance of finite-horizon agents decreases more when the horizon is short: short-horizon agents cannot learn rare transition and constantly change their policy when detecting a rare transition. This increases planning time as well (Figure 13, bottom). Consequently, the finite-horizon agent with a long horizon of $h = 30$ performs better than all of the other finite-horizon agents, although it performed poorly without this rare transition (Figure 12, top). Although we did not change the parameters of the multi-model agent, it performs better than all of the other agents. The multi-model agent learns accurate models of the environment, while being able to detect task changes. Other manipulations which necessitate that the agent learns an accurate model of the environment (such as adding reward uncertainty as in Figure 9) may produce similar results.

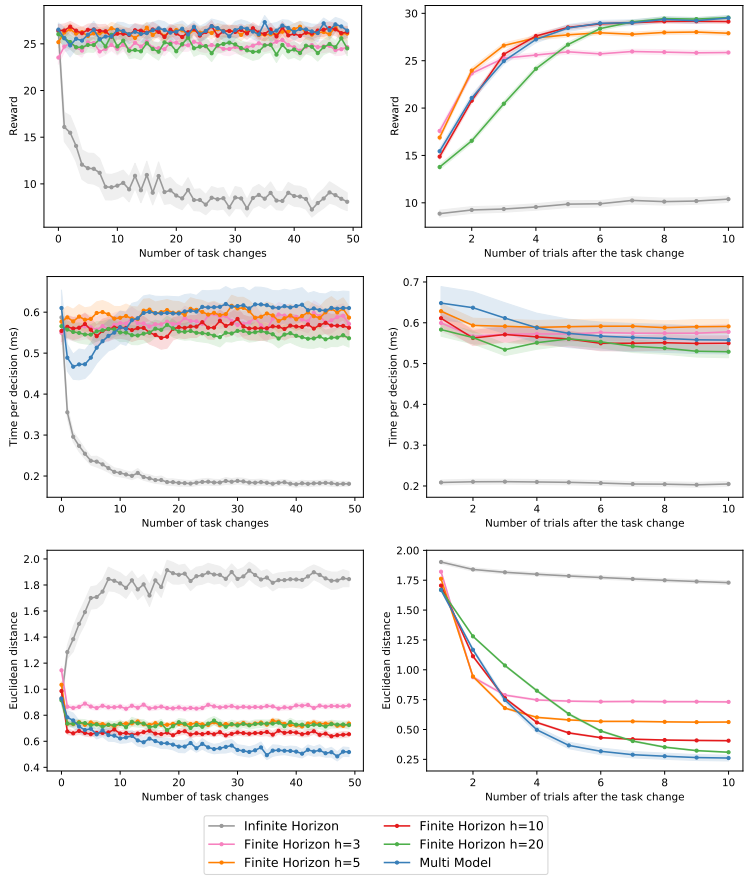


Figure 11: **Performance of the agents on the chain task.** The finite-horizon and the multi-model agents solve the task. All the agents are evaluated with three metrics: the reward accumulated (Top), the computational time (Middle) and the Euclidean distance to the true transitions of the environment (Bottom).

### 4.2 Parameter analysis of the multi-model agent

Thresholds and parameters can be sensitive to parameter fitting. In this section, we study whether the multi-model agent learns accurate models of the chain even when its parameters are wrongly specified. We ran experiments on the chain task because of its intermediate complexity. The multi-model agent uses the baseline parameters $h = 10$, $\Delta_c = 1$, $\Delta_m = 0.1$, $\max_{mod} = 5$ (Figure 11). We keep these parameters throughout the parameter analysis and change them one by one to see how they impact the multi-model agent behavior.

#### 4.2.1 Study horizon, model creation and change detection

The study horizon of the multi-model agent defines the maximal number of observations to evaluate the reliability of the models. If the horizon is small, the agent may not have enough information to decide whether it needs to change its current model or not. If the horizon is too long, changes may not be detected. Conversely, a short horizon emphasizes recent observations, potentially causing instability in model creation or model changes. Therefore, the horizon $h$ is highly connected to change detection. Another parameter which monitors change detection is the model creation threshold $\Delta_c$. If the model creation threshold is small, the agent might build unnecessary models. If it is high, the agent may not build enough models. The higher the model creation threshold, the closer the agent behaves like an infinite-horizon agent. In deterministic environments, the model creation threshold should be low to create a new model as soon as the agent observes a surprising event. In uncertain environments, the horizon should be high so that the agent samples enough information before deciding whether it should change models.

We compare the evolution of the number of models depending on the horizon $h$ and the model creation threshold $\Delta_c$ in Figure 14. We study the behavior of an agent with high model creation ($\Delta_c = 0.3$), an agent with low model creation ($\Delta_c = 1.5$), an agent with a short horizon ($h = 3$), an agent with a long horizon ($h = 20$) and an agent with a long

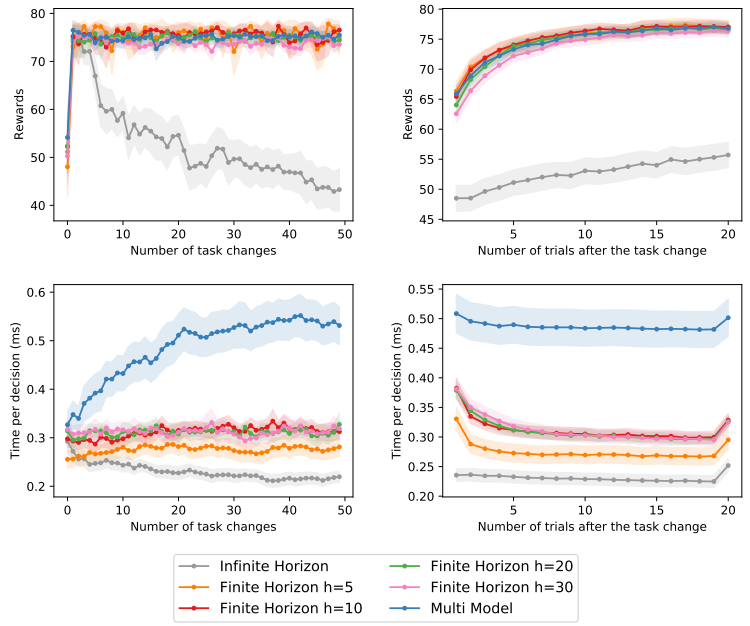


Figure 12: **Performance of the agents in the mazes**. (Top) The multi-model agent and the finite-horizon agents solve the task. (Bottom) The multi-model agent is more time demanding than other agents.

horizon but a high model creation ($h = 20,\ \Delta_c = 0.3$). For all the agents, we plot the total number of models, the number of models created, the number of models merged and the number of models forgotten over time.

The baseline agent creates 10 models and stabilizes to 20 total models, which corresponds to the true number of models (2 models for each of the 2 actions for 5 states). It does not need to merge nor forget any model. The agent with a high creation rate creates too many models. Nevertheless, the merging module compensates this by merging models which are similar. This limits the total number of models to 23 on the time scale studied. However, the number of models has not stabilized and the agent may further increase its number of models. The low creation agent does not create enough models and stabilizes around 18 models. The long-horizon agent creates models slower than the baseline agent, but almost reaches the appropriate number of models after 50 task changes. The small horizon agent creates a little too many models but stabilizes at 22 models. Finally, the agent with a long horizon and high model creation reaches and stabilizes to 20 models.

Although their parameters are incorrectly specified, all the agents created approximately 20 models (between 18 and 23). Yet, the horizon $h$ varies between $h = 3$ and $h = 20$ and the model creation threshold $\Delta_c$ varies between $\Delta_c = 0.3$ and $\Delta_c = 1.5$. Moreover, an agent with a long horizon $h = 20$ but a small model creation threshold $\Delta_c = 0.3$ creates the appropriate number of models. Finally, only the agents with high creation rates merged models together. These results indicate that the number of models the multi-model agent maintains is relatively stable to parameter changes.

The agents generally complete the chain task (Figure 15). The high-creation agent ($\Delta_c = 0.3$) receives approximately the same rewards as the baseline agent. Creating many models speeds up the adaptation to task changes but increases the sensitivity to surprising events. Additionally, creating models, merging them and arbitrating between them is computationally costly. Thus, the time complexity of the high creation agent increases over time. The low creation agent ($\Delta_c = 1.5$) does not create enough models. Therefore, the models it maintains are not accurate and the agent does not attain optimal performance. The short horizon agent ($h = 3$) is fast at detecting task changes but does not perform well in stable environments. The short horizon agent may be too sensitive to surprising events, changing models too frequently and not maintaining accurate models. The long horizon agent ($h = 20$) takes too long to detect changes, which reduces its performance on the task. However, lowering the model creation threshold improves change detection. The corresponding agent ($h = 20,\ \Delta_c = 0.3$) performs as well as the baseline agent.

An agent which creates too many models performs better than an agent which creates too little (Figure 15). A high model creation agent can merge similar models, whereas an agent which misses a task change permanently loses information. Creating a model and merging it quickly with a stored model sometimes out-speeds the switching system, which only gives the upper hand to a stored model if it explains the latest observations better than the current model.

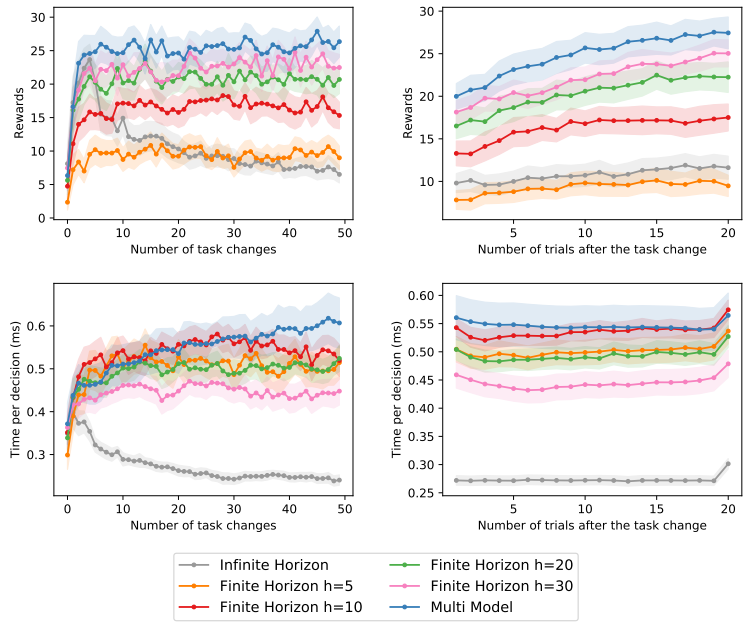


Figure 13: **Performance of the agents in the mazes with a rare transition to the initial state.** (Top) The multi-model agent outperforms the other agents, as it learns accurate models of the mazes for each task change. (Bottom) The computational costs of the finite-horizon agents increase in uncertain mazes.

#### 4.2.2 Model merging and model forgetting

To study how the multi-model agent merges or forgets models, we lower the threshold of model creation from $\Delta_c = 1$ to $\Delta_c = 0.3$ so that the agent creates too many models. We compare the evolution of the numbers of models depending on the merging threshold $\Delta_m$ and the maximal number of models $\max_{mod}$ in Figure 16. We study the behavior of an agent with high model merging ($\Delta_m = 0.5$), an agent with low model merging ($\Delta_m = 0.01$), an agent with no merging ($\Delta_m = 0$), and an agent with no merging and a low maximal number of models ($\Delta_m = 0$, $\max_{mod} = 2$).

The agent with a low merging threshold ($\Delta_m = 0.01$) does not merge enough models: the number of models does not stabilize and increases to more than 30 models. In contrast, the agent with a high merging threshold ($\Delta_m = 0.5$) merges too many models and does not maintain enough model diversity. We voluntarily chose parameters that would display these two behaviors. However, the merging threshold is rather stable. In this example, it varies of a factor 50, from $0.01$ to $0.5$. Interpreting the role of this parameter is straightforward: the lower it is, the less the agent merges models.

If the agent creates too many models and does not merge them, it can forget small models. Limiting the number of models may sometimes improve the performance of the agents: the agent with a maximal number of 2 models performs better than the one with 5 (Figure 17). The merging mechanism ensures that the agent does not waste time comparing similar models and improves their precision by merging them. Agents with low merging are the most time demanding (Figure 17, center). The agent with a high merging threshold performs well, although it does not learn the underlying transitions accurately. This illustrates that agents do not need precise models of the environment in the chain task to find the optimal policy.

#### 4.2.3 Conclusions on the role of each parameter

The horizon $h$ needs to be long enough to assess model reliability on enough observations. The horizon should not be too short, even in close-to-deterministic environments. Conversely, the agent can use long horizons in uncertain environments. If the horizon is too long, reducing the model creation threshold $\Delta_c$ improves the performance of the agent. The larger the model creation threshold $\Delta_c$, the more the agent performs like an infinite-horizon model. An agent with a low model creation threshold creates too many models, but the merging mechanism may limit the total number of models. The merging threshold $\Delta_m$ may not need a precise fit for each environment as it represents a distance between two distributions to merge. The maximal number of models $\max_{mod}$ acts as a safety net, so that the number of models does not increase dramatically when the creation and the merging parameters are incorrect. Overall, we demonstrated

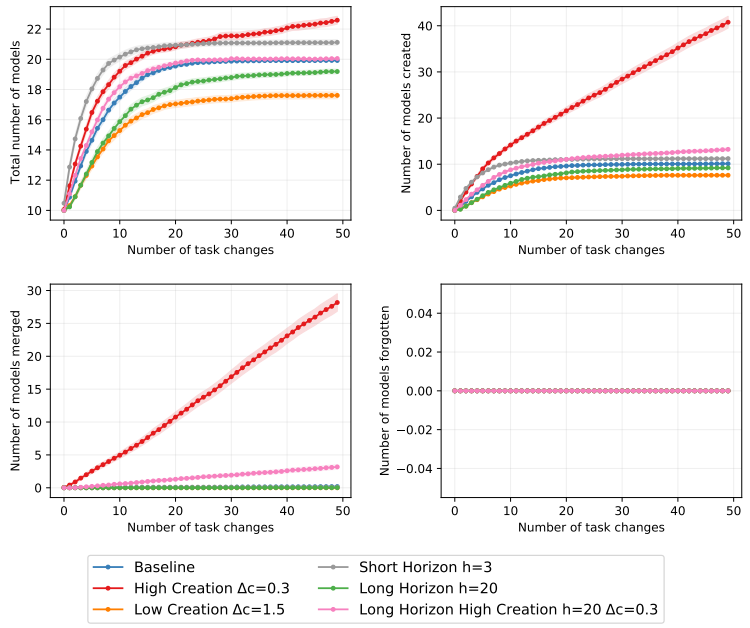


Figure 14: **Number of models depending on the horizon and the model creation threshold.** (Top-Left) Total number of models. (Top-Right) Number of models created. (Bottom-Left) Number of models merged. (Bottom-Right) Number of models forgotten.

that the multi-model agent performs well even with incorrect parameter values. Moreover, all the multi-model agents we tested were of comparable yet usually slightly larger computational cost than the baseline. The computational cost of the multi-model agent increases with the number of models.

# 5 Discussion

## 5.1 Conclusions

We introduced a new multi-model approach to deal with local task-changes in transitions, at a lower memory cost than context-based approaches (Figure 1). Based on this local-model-level approach, we presented a reinforcement learning agent which arbitrates between multiple local models (Figure 2). This agent can create, merge, forget models at the level of the state-action pair and can retrospectively detect when changes happen (Figures 3, 4 and 5). We compared the agent with infinite-horizon and finite-horizon agents in changing environments of various types and sizes (Figures 6 and 7). Our new method performs at least as well than the other single-model methods in all environments (Figures 8, 11, and 12). Essentially, our multi-model approach outperforms single-model methods in changing and uncertain environments (Figures 9, 10, and 13). This approach has explicable and stable parameters (Figures 14, 15, 16, and 17). The multi-model agent maintains infinite and accurate models of the environment and adapts to task changes which opens new ways to solve locally changing tasks. In future works, one can apply our method to detect changes in rewards, use model-novelty, model-reliability and co-variation of local models to improve the exploration and the performance of the multi-model agent, use the co-variation of local model changes to infer contextual changes, or test to what extent this multi-model approach can explain behavioral data.

## 5.2 Related work

Inferring contexts and adapting to them has recently been presented as a candidate to explain human behavior in many sub-fields of cognitive sciences, such as classical conditioning, episodic memory, instrumental learning, and motor adaptation [16]. In our work, we focus on decision-making and reinforcement learning, with an agent acting in a Markov decision process and trying to infer the current structure of the task to solve it.

At least two approaches from the literature tried to model how humans adapt to task changes: the PROBE algorithm [8] and the COIN algorithm [15]. Our multi-model approach is very different from the COIN algorithm which focuses on contextual inference to learn sensorimotor repertoires. Conversely, our approach shares some structural similarities

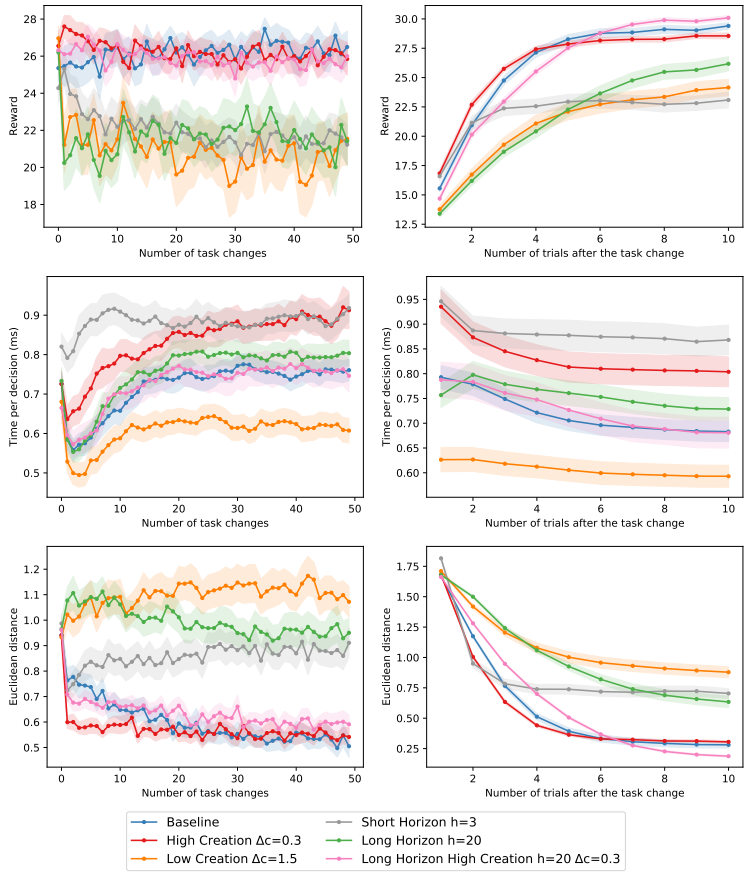


Figure 15: **Performance, times and distances to the true model for multi-model agents depending on the horizon and the model creation threshold.** (Top) Rewards accumulated. (Center) Computational times. (Bottom) Euclidean distance to the true transition model.

with the PROBE algorithm, which aims to model human behavior in a decision-making task. The PROBE approach builds different task-sets, arbitrates between them online based on their reliability and discards new models that are outperformed by stored models. At each time step, the reliability of all the task-sets is compared to the reliability of random behavior.

Our multi-model approach creates and arbitrates between different models, and uses a reliability measure, similar to the PROBE approach. However, there are several differences between the two methods. First, we do not work at the context level but at the transition model level. Second, our approach retrospectively detects the moment the switch happened, whereas the PROBE algorithm only adds the latest observation to the new context. Third, we do not compare our models to the reliability of random behavior. Instead, we use a sliding window for the latest observations and check if the current model explains them. Fourth, we use a merging module that can merge similar models, while there is no merging module in the PROBE approach. This merging module limits the number of models and our approach often does not forget models. Conversely, the PROBE method often discards unreliable newly created models which reduces its stability. Finally, our approach scales well to large environments and can be applied to any Markov Decision Process.

In the reinforcement learning literature, several algorithms tried to adapt to task changes with online arbitration between models. The Multiple Model-Based Reinforcement Learning (MMRL) algorithm [12] uses a responsibility signal to weigh the outputs of multiple specialist modules. This responsibility signal changes depending on how well each specialist predicts the next state. Contrary to our method, the MMRL algorithm uses a fixed number of models. To solve this limitation, the Reinforcement Learning with Context Detection (RL-CD) [10] method arbitrates between different contexts in Markov decision processes without knowing the number of contexts in advance. The algorithm uses an aggregate measure of discrepancy between the stored reward and transitions and the current experience. When no stored context can explain recent observations, it creates a new uniformly initialized context. As a result, the memory used by the RL-CD agent increases linearly with the number of contexts. More recent methods have either improved RL-CD

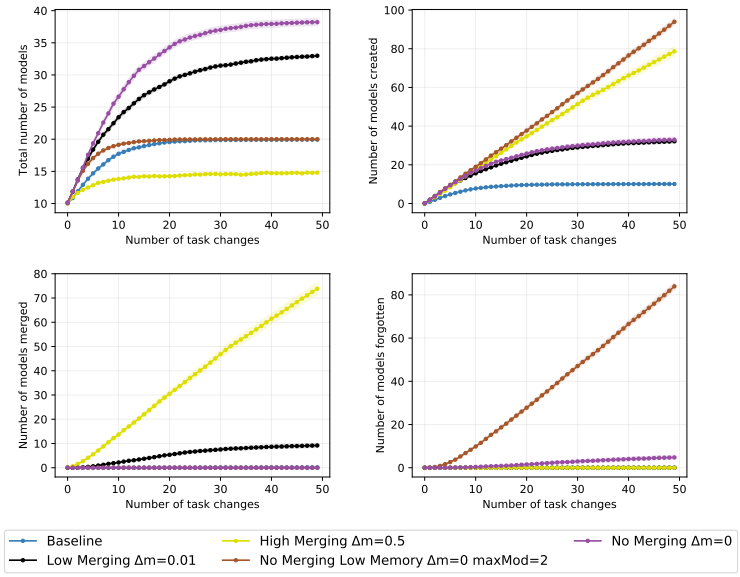


Figure 16: **Number of models depending on the model merging threshold and the maximal number of models.** (Top-Left) Total number of models. (Top-Right) Number of models created. (Bottom-Left) Number of models merged. (Bottom-Right) Number of models forgotten.

by refining the change detection method [14], demonstrated theoretical properties of a worst-case scenario algorithm using tree search when the environment changes gradually [18], or focused on detecting changes with a model-free approach [23]. However, these three algorithms use a context-level approach and not the local-model-level approach we presented (Figure 1). Thus, they would fail to adapt to the exponential number of contexts of the environments we introduced in this article.

### 5.3 Detecting changes in transitions and rewards

The agents we presented use a frequentist approach to compute the transition and reward functions. For transitions, this approach provides a distribution (an histogram) over arrival states. Conversely, the frequentist approach for the reward function only provides a point-estimate value (the mean of the reward distribution). To apply directly our approach to rewards, one should use a distributional approach (*e.g.*, a Bayesian approach with an initial normal prior). Using prior knowledge on the shape of the reward function would boost the performance of the task change detection, while reducing general applicability. To adapt to reward changes with no prior information, one can use an histogram approach close to the one we used for changes in transitions in this article.

If the agent tries to detect changes both in rewards and in transitions, it can combine both change detection metrics to detect joint changes. In the reinforcement learning literature, some algorithms of context detection use an aggregate measure of reward and transition changes to detect context changes [10]. An agent which detects joint changes in rewards and transitions can potentially adapt faster to changes when the two distributions correlate. However, the context detection methods would relearn the whole maps of transitions and of rewards, irrespective of their impact in the change detection. Adapting our local method to reward changes would allow to measure whether reward and transition distributions change together and update the models accordingly.

### 5.4 Dealing with drifting environments

In their everyday lives, animals sometimes face drifting environments with a small but steady change which becomes prominent over time. For example, the amount of water in a pond may decrease slightly over time based on weather conditions and the numbers of animals drinking out of it. The multi-model agent does not take into account drifting situations as it focuses on detecting abrupt local change. To consider drifting environments, the agent could use a finite but long horizon (of size $H >> h$), instead of an infinite horizon for the models it stores. The multi-model approach would remain similar to what was presented before, with this additional parameter.

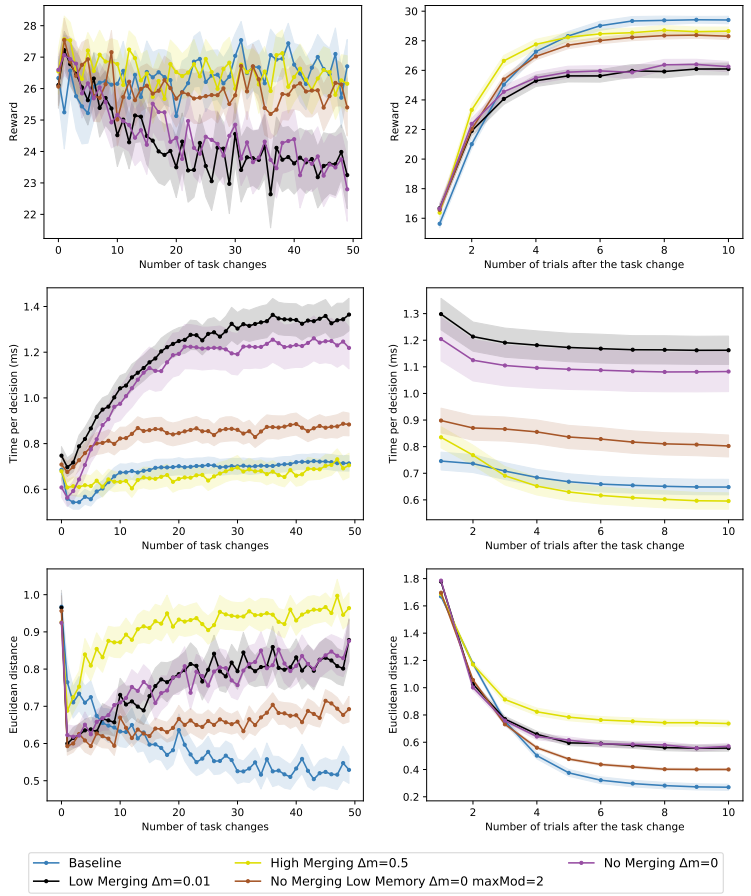


Figure 17: **Performance, times and distances to the true model for high creation multi-model agents depending on the merging threshold and the maximal number of models.** (Top) Rewards accumulated. (Center) Computational times. (Bottom) Euclidean distance to the true transition model.

### 5.5 Adding exploration to the multi-model agent

The multi-model approach we present in this article does not make any assumption on how the agent makes a decision. Hence, the multi-model agent can use many exploration methods to boost its performance. When creating a new model or when a model has little information, the agent may explore to gather information and add observations to the model. To implement this, the agent can directly use optimistic in the face of novelty methods which come from the count-based exploration literature for single-model agents [2]. These methods bias the behavior of the agent to gather more information on models which have too little observations (or counts).

The multi-model agent can also use model-reliability to favor exploration. When the Kullback-Leibler divergence of the current model increases, the reliability of the local model decreases. This means that there was a surprising observation that the agent needs to investigate. Boosting exploration towards surprising event may improve change detection. An important property of using model reliability as an incentive to explore is that this measure of curiosity eventually decreases as the agent samples more information. This avoids that the agent remains permanently motivated to explore something inherently uncertain which is a common limitation from the intrinsic motivation literature [4].

Finally, the multi-model agent can take advantage of the temporal co-variation of local changes to detect them faster. The agent could store a co-variation matrix and learn the associations between local changes with the retrospective change point detection mechanism. Once the agent learned this matrix and detects a change for a given state-action pair, it can re-explore all of the state-action pairs which correlate with it. This possibility to detect precisely when local changes happen would even allow to form contexts once the local changes are associated together. The main difference between context detection and the re-exploration of co-varying changes is that once the agent learned a context, it can infer the current state of the other local models directly.

## Acknowledgments

This research was funded in whole, or in part, by the French Agence Nationale de la Recherche (ANR) (ANR-21-CE33-0019-01 ELSA project). We would like to thank Olivier Serris for discussions on the change point detection method, and Abhishek Banerjee, Oriane Demandolx, Clarissa Montgomery and Matthias Tsai for their comments on a previous version of the manuscript.

## Declaration of interest

The authors declare no conflict of interest.

## Copyright

All figures are available on figshare under a CC BY 4.0 license.

## Code

The code is available online on github.

## References

[1] J Yu Angela and Peter Dayan. Uncertainty, neuromodulation, and attention. *Neuron*, 46(4):681–692, 2005.

[2] Ronen I Brafman and Moshe Tennenholtz. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3(Oct):213–231, 2002.

[3] Ken Caluwaerts, Mariacarla Staffa, Steve N'Guyen, Christophe Grand, Laurent Dollé, Antoine Favre-Félix, Benoît Girard, and Mehdi Khamassi. A biologically inspired meta-control navigation system for the psikharpax rat robot. *Bioinspiration & biomimetics*, 7(2):025009, 2012.

[4] Augustin Chartouny, Benoît Girard, and Mehdi Khamassi. Why learning progress needs absolute values: Comment on poli et al.(2024). *The European Journal of Neuroscience*, 61(1):e16635, 2024.

[5] Augustin Chartouny, Jean-Pierre Nadal, and Mehdi Khamassi. [∼Re] Exploration in Model-based Reinforcement Learning by Empirically Estimating Learning Progress. *The ReScience journal*, 73, September 2024.

[6] Samuel Choi, Dit-Yan Yeung, and Nevin Zhang. An environment model for nonstationary reinforcement learning. *Advances in neural information processing systems*, 12, 1999.

[7] François Cinotti, Virginie Fresno, Nassim Aklil, Etienne Coutureau, Benoît Girard, Alain R Marchand, and Mehdi Khamassi. Dopamine blockade impairs the exploration-exploitation trade-off in rats. *Scientific reports*, 9(1):6770, 2019.

[8] Anne Collins and Etienne Koechlin. Reasoning, learning, and creativity: frontal lobe function and human decision-making. *PLoS biology*, 10(3):e1001293, 2012.

[9] Jukka Corander, Ulpu Remes, and Timo Koski. On the jensen-shannon divergence and the variation distance for categorical probability distributions. *Kybernetika*, page 879–907, January 2022.

[10] Bruno C Da Silva, Eduardo W Basso, Ana LC Bazzan, and Paulo M Engel. Dealing with non-stationary environments using context detection. In *Proceedings of the 23rd international conference on Machine learning*, pages 217–224, 2006.

[11] Richard Dearden, Nir Friedman, and Stuart Russell. Bayesian q-learning. *Aaai/iaai*, 1998:761–768, 1998.

[12] Kenji Doya, Kazuyuki Samejima, Ken-ichi Katagiri, and Mitsuo Kawato. Multiple model-based reinforcement learning. *Neural computation*, 14(6):1347–1369, 2002.

[13] Samuel J Gershman, David M Blei, and Yael Niv. Context, learning, and extinction. *Psychological review*, 117(1):197, 2010.

[14] Emmanuel Hadoux, Aurélie Beynier, and Paul Weng. Sequential decision-making under non-stationary environments via sequential change-point detection. In *Learning over multiple contexts (LMCE)*, 2014.

[15] James B Heald, Máté Lengyel, and Daniel M Wolpert. Contextual inference underlies the learning of sensorimotor repertoires. *Nature*, 600(7889):489–493, 2021.

[16] James B Heald, Máté Lengyel, and Daniel M Wolpert. Contextual inference in learning and memory. *Trends in cognitive sciences*, 27(1):43–64, 2023.

[17] Khimya Khetarpal, Matthew Riemer, Irina Rish, and Doina Precup. Towards continual reinforcement learning: A review and perspectives. *Journal of Artificial Intelligence Research*, 75:1401–1476, 2022.

[18] Erwan Lecarpentier and Emmanuel Rachelson. Non-stationary markov decision processes, a worst-case approach using model-based reinforcement learning. *Advances in neural information processing systems*, 32, 2019.

[19] Manuel Lopes, Tobias Lang, Marc Toussaint, and Pierre-Yves Oudeyer. Exploration in model-based reinforcement learning by empirically estimating learning progress. *Advances in neural information processing systems*, 25, 2012.

[20] Louis-Emmanuel Martinet, Denis Sheynikhovich, Karim Benchenane, and Angelo Arleo. Spatial learning and action planning in a prefrontal cortical network model. *PLoS computational biology*, 7(5):e1002045, 2011.

[21] Stephen Monsell. Task switching. *Trends in cognitive sciences*, 7(3):134–140, 2003.

[22] Rani Moran, Mehdi Keramati, Peter Dayan, and Raymond J Dolan. Retrospective model-based inference guides model-free credit assignment. *Nature communications*, 10(1):750, 2019.

[23] Sindhu Padakandla, Prabuchandran KJ, and Shalabh Bhatnagar. Reinforcement learning algorithm for non-stationary environments. *Applied Intelligence*, 50(11):3590–3606, 2020.

[24] Pascal Poupart, Nikos Vlassis, Jesse Hoey, and Kevin Regan. An analytic solution to discrete bayesian reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, pages 697–704, 2006.

[25] Ronald A Rensink. Change detection. *Annual review of psychology*, 53(1):245–277, 2002.

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

[27] Qingyun Wu, Naveen Iyer, and Hongning Wang. Learning contextual bandits in a non-stationary environment. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 495–504, 2018.

# A Additional figures

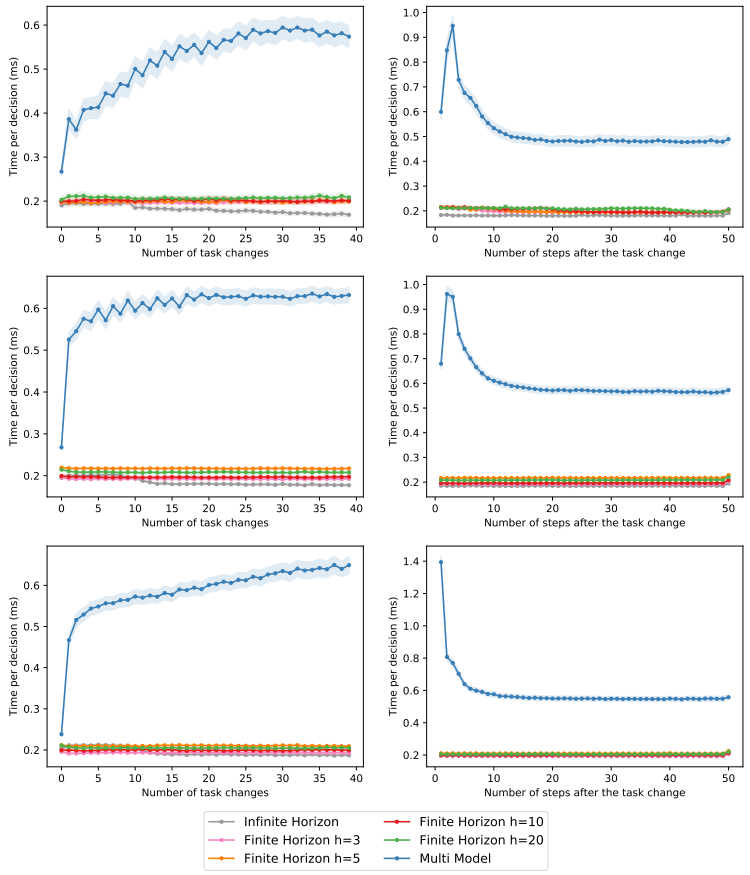


Figure 18: **Computational times of the agents the three-state task.** (Top) Classical task. (Center) Task with reward uncertainty. (Bottom) Task with transition uncertainty. In all three cases, the multi-model is more computationally expensive than other methods. For the multi-model agent, computational time increases right after the detection of a task change before decreasing back to a smaller value after a few steps.

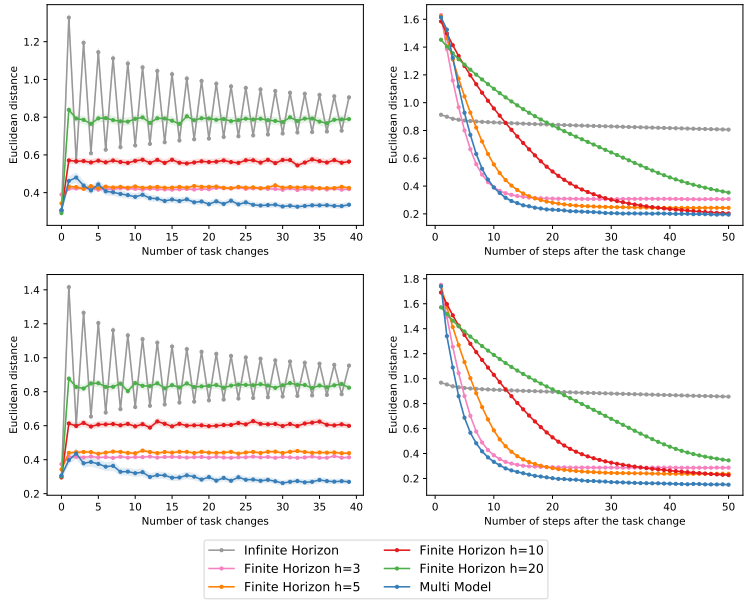


Figure 19: **Euclidean distance for the agents on the three-state task.** (Top) Task with reward uncertainty. (Bottom) Task with transition uncertainty. The multi-model agent has a more accurate model of the transitions than all of the other agents.

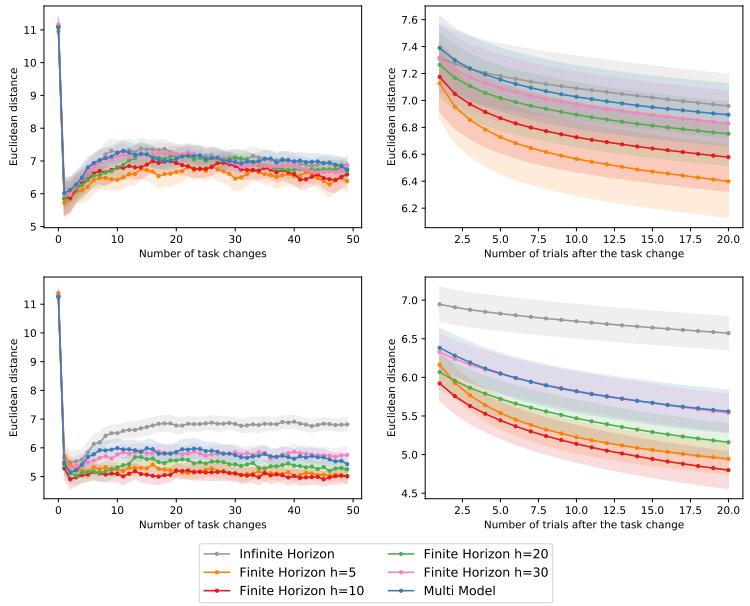


Figure 20: **Euclidean distance for the agents in the mazes.** (Top) Classical task. (Bottom) Task with transition uncertainty. It is hard to interpret these results, as many agents do not explore all of the state-action pairs and thus have erroneous models of the environment.