Long-Horizon Planning with Predictable Skills

Nico Gürtler **Georg Martius**

University of Tübingen and Max Planck Institute for Intelligent Systems

Abstract

Model-based reinforcement learning (RL) leverages learned world models to plan ahead or train in imagination. Recently, this approach has significantly improved sample efficiency and performance across various challenging domains ranging from playing games to controlling robots. However, there are fundamental limits to how accurate the long-term predictions of a world model can be, for example due to unstable environment dynamics or partial observability. These issues are further exacerbated by the compounding error problem. Model-based RL is therefore generally limited to short rollouts with the world model, and consequently struggles with long-term credit assignment. We argue that this limitation can be addressed by modeling the outcome of temporally extended skills instead of the effect of primitive actions. To this end, we propose a mutual-information-based skill learning objective that ensures predictable, diverse, and task-related behavior. The resulting skills compensate for perturbations and drifts, enabling stable long-horizon planning. We design a sample-efficient hierarchical agent consisting of model predictive control with an abstract skill world model on the higher level, and skill execution on the lower level. We demonstrate that our algorithm, *Stable Planning with Temporally Extended Skills (SPlaTES)*, solves a range of challenging long-horizon continuous control problems, outperforming competitive model-based and skill-based methods.[1]

1 Introduction

Learning a world model is a promising route to obtaining competent and versatile agents. In many environments, learning approximate dynamics is fast and enables a shift from trial and error to more targeted problem solving via planning or learning from synthetic experience. Consequently, model-based reinforcement learning (RL) recently reached unprecedented sample efficiency and asymptotic performance in many challenging domains (Schrittwieser et al., 2020; Hafner et al., 2021; Hansen et al., 2024). However, due to compounding model errors, these methods are generally limited to rolling out the model for a small number of time steps. This severely reduces their ability to solve complex tasks that require longer rollout horizons to discover good solutions. Unfortunately, improving the accuracy of the world model is costly and quickly leads to diminishing returns, in particular in environments with stochastic or unstable dynamics.

Alternatively, short model rollouts can be complemented with a learned value function to capture the impact of actions on future rewards. Although such hybrid methods have achieved remarkable results on hard *long-horizon* tasks (Hafner et al., 2023), their ability to perform *long-term credit assignment* is still limited: Learning from short model rollouts requires temporal difference (TD) learning, which can become unstable for the high discount factors required for discovering non-myopic behavior. Undesirable artifacts in learned value functions can also impede progress (Bagatella & Martius, 2023). Moreover, the problematic combination of function approximation, bootstrapping

[1] Code and videos can be found on the project page.

18th European Workshop on Reinforcement Learning (EWRL 2025).

and off-policy training, known as the 'deadly triad' (Sutton et al., 1998), is at the center of many modern model-based frameworks (Hansen et al., 2024). Hence, solving long-term credit assignment efficiently remains an open challenge for model-based RL methods.

Interestingly, humans excel at long-horizon planning despite our inaccurate short-term predictions. The key to this remarkable ability is abstraction: rather than planning fine-grained movements, we usually leverage high-level skills to solve complex problems. For example, when boiling a pack of pasta, we can rely on our hands to open the tap to fill water into the pot, to put the pot on the stove, to turn on the stove, and so on, all without having to worry about detailed movements. Crucially, even unforeseen events, such as the pot beginning to slip from our hand, have little to no impact on the overall outcome as we automatically readjust our grip. Hence, thinking in terms of predictable skills instead of primitive actions replaces unstable or stochastic environment dynamics with a more stable abstract world model, suitable for long-horizon planning.

We propose to replicate this strategy by learning temporally extended skills alongside an abstract world model that predicts skill outcomes. These outcomes should be diverse for different skills, but predictable for each individual skill. We therefore maximize the mutual information between the transition induced by the skill and the skill vector (Eysenbach et al., 2019). Concretely, we train the skills with RL using an approximation to the mutual information derived from the abstract world model (Sharma et al., 2020b). Since each skill lasts for multiple time steps, it can detect and counteract errors in its trajectory. Intuitively, by training the skills to realize what the world model predicts and the world model to predict what the skills achieve, stable high-level dynamics are obtained. Our proposed method *Stable Planning with Temporally Extended Skills (SPlaTES)* consists of (i) learning the temporally extended skills in tandem with the abstract world model, while (ii) using them for model predictive control (MPC).

In high-dimensional environments, in particular, it is crucial that the learned skills focus on what matters for the task. Most hierarchical RL methods (Sutton et al., 1999) that tackle long-horizon tasks use domain knowledge to define a subgoal or skill space that captures task-relevant parts of the state (Eysenbach et al., 2019; Nachum et al., 2018; Levy et al., 2019; Sharma et al., 2020b). We show that learning a low-dimensional abstract latent space by fitting the reward and propagating gradients back to the encoder is possible for sufficiently dense rewards. By learning skills that control the transitions in this space, we obtain task-related behavior. Hence, we do not need access to a handcrafted latent space or reward function.

Our contributions are: (i) We propose SPlaTES, a sample-efficient hierarchical RL method that learns temporally extended skills on the lower level, and an abstract world model over skill outcomes on the higher level. Both levels are model-based and perform MPC on different timescales. (ii) We show that our skill learning objective yields diverse, predictable, task-related, and error-correcting behavior. (iii) Planning over entire episodes enables SPlaTES to outperform competitive model-based and skill-based methods on a range of challenging long-horizon continuous control tasks.

2 Preliminaries

In reinforcement learning (RL), an agent is trained to maximize the cumulative reward through interactions with the environment. The environment can be formalized as a Markov Decision Process (MDP), $\mathcal{M} = (\mathcal{S}, \mathcal{A}, p, r, \rho_0, \gamma)$, where $\mathcal{S}$ denotes the state space, $\mathcal{A}$ the action space, $p\left(s' \mid s, a\right)$ the probability (density) of transitioning from state s into s' when choosing action a, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ the reward function, ρ_0 the distribution of the initial state, and $\gamma \in [0, 1)$ the discount factor. In the infinite horizon setting, the RL objective is the maximization of the expected discounted return $G^\pi = \mathbb{E}_{a_t \sim \pi(\cdot|s_t), s_0 \sim \rho_0}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right]$ when following a policy $\pi : \mathcal{S} \to \Delta\left(\mathcal{A}\right)$.

In *model-based* RL, the agent learns a dynamics model $\hat{p}\left(s' \mid s, a\right)$ and a reward model $\hat{r}\left(s, a\right)$ from transitions (s, a, r, s'). The combination of $\hat{p}$ and $\hat{r}$ yields a world model, which can be used to perform rollouts in imagination by repeatedly sampling from the transition probability. At time step t, define $\hat{s}_t = s_t$. For a horizon H and a fixed action sequence $a_t, \ldots, a_{t+H}$, the recursive relation $\hat{s}_{t+k+1} \sim \hat{p}\left(\cdot \mid \hat{s}_{t+k}, a_{t+k}\right)$ defines a model rollout, which may be used with any model-free algorithm to train a policy. Alternatively, the world model may be used directly for planning by optimizing the return-to-go $\sum_{k=0}^{H} \hat{r}(\hat{s}_{t+k}, a_{t+k})$ with respect to the action sequence. We use iCEM (Pinneri et al., 2021) for this purpose, a zero-order optimization method adapted for continuous

control. By replanning after each time step in a model predictive control (MPC) fashion, errors from imperfect model learning or optimization can be mitigated to some extent.

3 Challenges in long-horizon predictions

Taking the same sequence of actions in the environment and in a world model usually leads to trajectories that deviate after a small number of time steps. There are several reasons for this mismatch:

(i) Approximation errors in the dynamics model may result in a small error ϵ after every time step, even in deterministic environments, $\hat{s} = \hat{p}(s, a) = p(s, a) + \epsilon$. **(ii) Partial observability** denies the world model access to the full, accurate state of the environment. This may be due to missing or noisy measurements. **(iii) Stochasticity** inherent to the environment renders the transitions non-deterministic. **(iv) Unstable dynamics** (Slotine et al., 1991) prevent already present errors from shrinking or can even increase them over time. In chaotic environments (Ott, 2002), even small errors in the initial state can lead to large discrepancies after a short period of time.

Small errors from these sources can add up when repeatedly applying the learned dynamics model, leading to the *compounding error problem* (Lambert et al., 2022). Hence, long-term predictions in many complex environments are infeasible in practice. Even the predictions of an accurate world model will quickly diverge from the true dynamics in such a system (see Figure 1). Trying to circumvent the compounding error problem by directly predicting several steps ahead can improve the accuracy in certain environments (Neitz et al., 2018; Lambert et al., 2021) but cannot solve the fundamental problem of predicting outcomes in unstable systems.

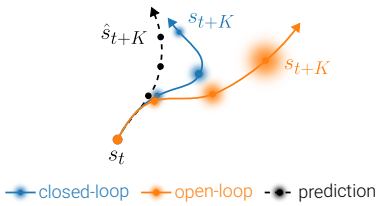


Figure 1: Taking a fixed sequence of actions in the real environment (open-loop in orange) and in the world model (prediction, dashed in black) results in trajectories that quickly diverge. A stable low-level policy (or skill) can compensate perturbations and ensure that prediction and reality stay close to each other (closed-loop in blue).

4 Method

Taking inspiration from how humans solve long-horizon problems, we propose to plan over temporally extended skills instead of primitive actions. While the environment as such may have undesirable properties like unstable or stochastic dynamics, suitable skills can mitigate these. We therefore aim to construct a planning-friendly abstraction of the environment by learning low-level skill policies that are trained to "achieve what an abstract world model predicts". By training such skills in tandem with an abstract world model, long-horizon planning becomes feasible. The rest of this section introduces our method Stable Planning with Temporally Extended Skills (SPlaTES).

4.1 Abstract POMDP

We first define an abstraction of the environment that is more suitable for long-horizon planning than the original MDP. To discard expendable details in the MDP, we adopt the form of a partially observable MDP (POMDP) (Kaelbling et al., 1998), which operates on a coarser timescale. This POMDP $\bar{\mathcal{M}} = \left(\mathcal{S}, \bar{\mathcal{A}}, \bar{p}_\pi, \bar{r}_\pi, \rho_0, \bar{\mathcal{S}}, f, \bar{\gamma}\right)$ inherits the state space $\mathcal{S}$ and the initial state distribution ρ_0 from the original MDP $\mathcal{M}$.

To implement *temporal abstraction*, we transfer control to a skill for $K \in \mathbb{N}^+$ time steps. During this time interval, the skill policy chooses primitive actions. We furthermore identify a skill by a skill vector $\bar{a} \in \bar{\mathcal{A}} = [-1, 1]^{d_{\bar{\mathcal{A}}}}$ with $d_{\bar{\mathcal{A}}} \in \mathbb{N}_+$. Choosing a continuous skill representation allows for smooth interpolation between behaviors which has advantages over discrete skills in domains such as manipulation or locomotion (Sharma et al., 2020b). The dynamics $\bar{p}_\pi\left(s_{t+K} \mid s_t, \bar{a}\right)$ of the POMDP are thus induced by the skill policy $\pi(a_t \mid s_t, \ldots, \bar{a})$ and the environment dynamics $p\left(s_{t+1} \mid s_t, a_t\right)$. The reward along a skill execution is accumulated, $\bar{r}_\pi = \sum_{n=t}^{t+K} r(s_n, a_n)$. Our

skills can be considered as a continuous version of options (Sutton et al., 1999) with a fixed temporal extent.

To facilitate planning, we map environment states to a more compact representation acting as an information bottleneck. As this potentially discards information, we may lose the Markov property and technically obtain only *observations* of the environment. Nevertheless, we refer to these representations as *abstract states*, as they are used for planning. In Section B.4, we discuss how our skill-learning objective can mitigate partial observability and approximately restore the Markov property. To ensure that the encoder $f : \mathcal{S} \to \bar{\mathcal{S}}$ focuses on task-relevant parts of the state, we train a reward function on the abstract state space $\bar{\mathcal{S}}$ and propagate gradients back to the encoder. Clearly, this requires the reward $\bar{r}_\pi$ to be sufficiently dense. We furthermore choose a low-capacity encoder to ensure that it is selecting aspects of the state rather than pre-computing the reward. Crucially, this provides us with a space that is sufficiently low-dimensional for a diversity-maximizing skill learning objective, as detailed in Section 4.3.

4.2 Abstract world model

The abstract world model $\hat{\bar{\mathcal{M}}}$ is trained to predict the next abstract state $\bar{s}' = f\left(s_{t+K}\right)$, as well as the reward $\bar{r}_\pi$ accumulated during the K steps of executing a skill.

The skill policy and partial observability introduce stochasticity to the transitions of the POMDP $\bar{\mathcal{M}}$, even if the original MDP $\mathcal{M}$ is deterministic. As K steps elapse in the environment during one abstract step, and due to partial observability, the distribution of $\bar{s}'$ can furthermore become multimodal. To take stochasticity and multimodality into account, we realize the abstract dynamics model $\hat{\bar{p}}\left(\bar{s}'|\bar{s}, \bar{a}\right)$ as a mixture of Gaussians, similar to Sharma et al. (2020b). Learning a distribution over the next abstract state additionally allows for taking chaotic or highly unstable dynamics into account that are too complex to be modeled precisely.

Note that the POMDP $\bar{\mathcal{M}}$ is non-stationary, since the skill policy π is changing during training. The abstract world model therefore has to track these changes. Moreover, abstract transitions $(\bar{s}, \bar{a}, \bar{r}, \bar{s}')$ become outdated quickly. Together with the temporal abstraction, this implies that the data for learning the abstract world model is very limited. In principle, importance sampling (IS) could be applied to outdated transitions if the skill policy is known. However, since the importance weights would correspond to a product of all K action probabilities in one skill execution, they in practice become very small. We therefore found that IS does not help, similar to Shi et al. (2023), and choose a sufficiently small buffer size to avoid outdated transitions.

4.3 Skill learning

The abstract actions $\bar{a}$ identify **temporally extended** skills. To make long-term planning in $\bar{\mathcal{M}}$ successful, these skills should (i) lead to **predictable** outcomes, (ii) be **useful** for the considered family of RL problems, and (iii) be **diverse**, i.e. allow for flexible control of the underlying MDP.

Similar to Dynamics-Aware Unsupervised Discovery of Skills (DADS) (Sharma et al., 2020b), we choose to implement predictability and diversity with a mutual-information-based objective. More concretely, we define the skill learning objective as the maximization of the mutual information of the skill vector $\bar{a}$ and the abstract state $\bar{s}' = f\left(s_{t+K}\right)$ at termination of the skill. Since we are interested in controlling the transitions in the abstract POMDP, we condition the skill on the abstract state $\bar{s} = f\left(s_t\right)$. This yields the following optimization objective for the skill policy π:

$$\pi^* = \arg\max_\pi I(\bar{s}'; \bar{a} \mid \bar{s}) = \arg\max_\pi H(\bar{s}' \mid \bar{s}) - H(\bar{s}' \mid \bar{s}, \bar{a}) \,. \tag{1}$$

Intuitively, $I(\bar{s}'; \bar{a} \mid \bar{s})$ corresponds to the amount of information that is revealed about the next abstract state when being presented with the skill vector. It is instructive to split the mutual information into a difference of two terms (see RHS of Equation 1): Maximizing the entropy of the next state $\bar{s}'$ conditioned on the current state $\bar{s}$ makes sure the skills can realize a diverse set of transitions in the abstract state space. Minimizing the entropy of $\bar{s}'$ conditioned on $\bar{s}$ *and the skill vector* $\bar{a}$ ensures that a specific skill reaches a predictable next state.

Evaluating $I(\bar{s}'; \bar{a} \mid \bar{s})$ poses two challenges: Firstly, it requires access to the abstract dynamics $\bar{p}\left(\bar{s}'|\bar{s}, \bar{a}\right)$, and secondly, it involves integration over realizations of the random variable $\bar{a}$. Following Sharma et al. (2020b), we tackle the first issue by approximating the dynamics with our world

model $\hat{\bar{p}}\left(\bar{s}'|\bar{s},\bar{a}\right)$, and the second by Monte Carlo sampling. A detailed derivation of the resulting approximation

$$I(\bar{s}';\bar{a}\mid\bar{s})\approx\mathbb{E}_{\bar{s},\bar{a},\bar{s}'}\left[\phi(\bar{s}';\bar{s},\bar{a})\right] \tag{2}$$

with the potential

$$\phi(\bar{s}';\bar{s},\bar{a})\coloneqq\log\frac{\hat{\bar{p}}\left(\bar{s}'\mid\bar{s},\bar{a}\right)}{\frac{1}{N}\sum_{i=1}^{N}\hat{\bar{p}}\left(\bar{s}'\mid\bar{s},\bar{a}_i\right)}\ ,\ \ \bar{a}_i\sim\mathcal{U}(\bar{A})\ , \tag{3}$$

can be found in subsection B.1.

To learn the skills, we maximize $\mathbb{E}_{\bar{s},\bar{a},\bar{s}'}\left[\phi(\bar{s}';\bar{s},\bar{a})\right]$ with RL. To this end, the potential difference between consecutive time steps serves as a dense reward for every intra-skill time step $k\in[0,\ldots,K-1]$,

$$r_{\text{skill}}(s_{t+k},a,s_{t+k+1};\bar{s},\bar{a})\coloneqq\phi(f(s_{t+k+1});\bar{s},\bar{a})-\phi(f(s_{t+k});\bar{s},\bar{a})\ . \tag{4}$$

We use TD-MPC2 (Hansen et al., 2024), a model-based RL method for continuous control that incorporates short-horizon planning, to learn the skill policy. In preliminary tests, it achieved higher sample efficiency than model-free algorithms. When sampling a batch from the replay buffer, we recompute the skill learning reward as it changes when the abstract world model gets updated. Furthermore, we slightly modify TD-MPC2 to keep track of k, $\bar{s}$, and $\bar{a}$ when rolling out its model.

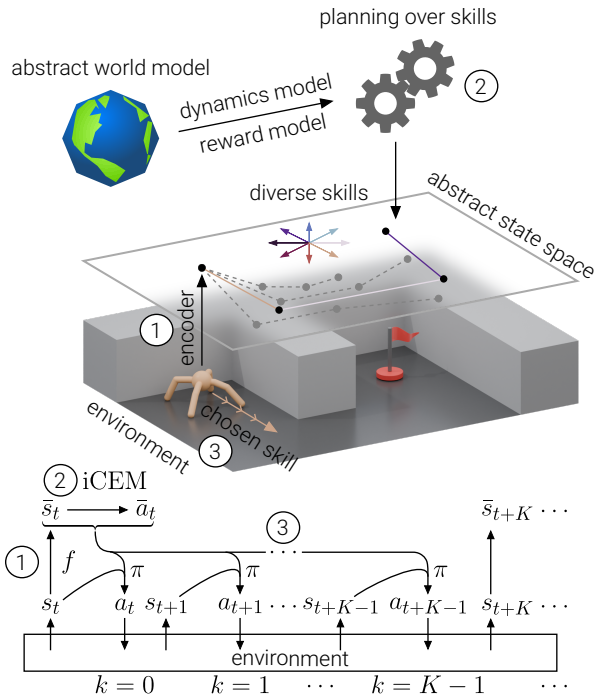
Crucially, the reward in Equation 4 encourages skills to be predictable relative to where they started and consequently depends on $\bar{s}$. This requires the skill policy to be conditioned on $\bar{s}$ as well. Since the skill execution has a finite horizon K, we additionally condition on the intra-skill step k (Pardo et al., 2018). Hence, the skill policy has the form $\pi\left(a_{t+k}\mid s_{t+k},k,\bar{s},\bar{a}\right)$. This conditioning allows the skills to *compensate perturbations and drifts* as they can detect any mismatch between where they are after k steps relative to $\bar{s}$ and where they intend to be (see Figure 1). This ability distinguishes our temporally extended skills from the memoryless skills DADS learns, and greatly contributes to the stability of our abstract world model. To ensure that the skills can be successfully chained, we furthermore bootstrap when the skill vector changes. As we want all possible transitions between skills to work, we replace the Q-function in the TD-target with a version trained to fit the expected Q-value over all possible next skills in this case (see Section B.6).

Applying common diversity-maximizing skill learning objectives directly in complex environments usually leads to behaviors that are not task-related. A common fix is the manual definition of a subspace that captures what is essential for the task, e.g. the x-y coordinate of an agent in a maze (Eysenbach et al., 2019; Sharma et al., 2020b). By instead learning the encoder based on reward and value prediction, we are more flexible and less reliant on domain knowledge, while still ensuring that the learned skills are useful for the (family of) tasks we are interested in.

4.4 The hierarchical agent: Abstract planning over task-related skills

Equipped with suitable skills and an abstract world model, we can now construct a hierarchical agent that solves the RL problem. To this end, we perform model predictive control (MPC) using the iCEM method with the abstract world model on the higher level and execute the skill policy on the lower level. Intuitively, the higher level breaks the task down into a sequence of skills, while the lower level executes them. We train all elements of the hierarchy online and in tandem, meaning that the abstract world model shapes the skill reward via Equation 4 and the behavior of the skill policy in turn determines the abstract world model. Hence, the skills are trained to fit the abstract world model and vice versa. Figure 2 presents an overview of the algorithm.

Training the world model of the POMDP outlined above automatically inherits its abstractions. This provides several benefits for planning: Firstly, the required MPC horizon is reduced by a factor of K (the length of one skill execution), and the state and action spaces are low dimensional. Combined with the stability of the skills, planning over whole episodes instead of a small number of time steps becomes feasible. Secondly, as replanning the sequence of skills only happens every K steps, the computational cost of MPC is reduced by a factor of K^2 compared to planning over the same effective horizon with a low-level model. Furthermore, the skills are trained to efficiently move through the abstract state space which encodes everything that is crucial for the task. Together with temporal abstraction, this greatly enhances exploration and automatically focuses it on the task (as conveyed by the reward function).



Algorithm 1: SPlaTES

$t \leftarrow 0;\ s \leftarrow s_0$
while *training* **do**
  $\bar{s} \leftarrow f(s);\ \bar{r} = 0$ ①
  $\bar{a} \leftarrow \text{iCEM}(\bar{s}; \hat{\mathcal{M}})[0]$ ②
  for $k = 0$ **to** K **do** ③
    $a \sim \pi\left(\cdot \mid s, k, \bar{s}, \bar{a}\right)$
    $s', r_{\text{env}} \leftarrow \text{env.step}(a)$
    $r_{\text{skill}} \leftarrow \phi(f(s'); \bar{s}, \bar{a})$
      $\quad - \phi(f(s); \bar{s}, \bar{a})$
    TDMPC2 $\leftarrow (s, a, r_{\text{skill}}, s')$
    TDMPC2.update()
    $\bar{r} \leftarrow \bar{r} + r_{\text{env}};\ s \leftarrow s'$
  end
  $\hat{\mathcal{M}} \leftarrow (\bar{s}, \bar{a}, \bar{r}, f(s))$
  $\hat{\mathcal{M}}$.update()
end

Figure 2: Left: Our proposed hierarchical algorithm SPlaTES in three steps; ① Encode the environment state to obtain an abstract state. ② Plan (with iCEM) a skill sequence starting from the abstract state, which maximizes the episode return. ③ Execute the first skill in the sequence for K steps. Do high-level MPC by starting from ① again. Right: Pseudocode for SPlaTES. The skill policy π denotes the action distribution induced by TD-MPC2 with MPC enabled.

Moreover, the hierarchical agent is able to correct its mistakes on two time scales: Firstly, small perturbations can be compensated by reflex-like behavior of the skills on the lower-level (think stumbling or something slipping from the agent's grasp). Secondly, MPC will automatically adjust the plan for the rest of the episode in a more deliberate way should the agent deviate further from the intended path. We provide further implementation details in Section B.6.

5 Experiments

We aim to answer the following questions with our empirical evaluation of SPlaTES:

1. Are the learned skills predictable, diverse, and useful for the task?
2. How does SPlaTES compare to existing methods in terms of sample-efficiency and asymptotic performance, in particular on challenging long-horizon tasks?
3. Can we distill the hierarchical SPlaTES model into a flat TD-MPC2 agent?

We compare to three baselines that cover model-based RL, hindsight relabeling, and skill discovery:

TD-MPC2 (with and without HER) (Hansen et al., 2024), a flat model-based RL method tailored to continuous control that achieves state-of-the-art sample efficiency by combining MPC with a learned policy and Q-function. We additionally combine TD-MPC2 with Hindsight Experience Replay (HER) to test if hindsight enables the agent to escape local optima.

DADS + MPC (Sharma et al., 2020b), a skill discovery method based on mutual information estimation with a reward similar to Equation 3. Unlike SPlaTES, DADS does not implement temporal abstraction, i.e., it tries to learn skills that control atomic transitions. DADS furthermore requires privileged information in the form of a projection to a compact latent space that encodes what matters for the task, e.g., the x-y coordinates for locomotion. To decouple the impact of the skill learning objective from the underlying RL algorithm, we implement a TD-MPC2-based version of DADS and use the same MPC code as for SPlaTES for choosing skills (see Appendix C for details).

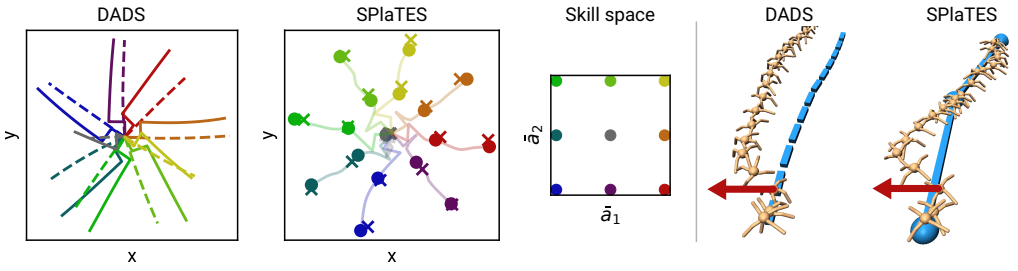


Figure 3: **Left: Skills applied to a velocity-controlled point mass under a perturbation.** DADS: Predictions are marked by dashed lines. SPlaTES: Predictions of the abstract world model are marked by crosses, actual states at the end of skill executions by circles. SPlaTES compensates the perturbation and stays close to the prediction while DADS cannot recover from it. **Right: Execution of a fixed skill sequence with a quadruped.** A force is applied in one time step (red arrow). SPlaTES corrects the resulting deviation while DADS cannot.

Our empirical evaluations are conducted in different variations of the following environments:

Fetch Pick & Place (Plappert et al., 2018): A robot arm with a two-fingered parallel gripper manipulating a block on a desk. The action specifies a desired displacement of the end effector and gripper fingers. To make the task harder, we added a variant in which the robot has to lift the block over a barrier to reach the goal (**Fetch P&P Barrier**).

Ant Maze (Fu et al., 2020): A torque-controlled quadruped navigating a maze. Long-term credit assignment is critical in this environment as going around a wall sometimes temporarily increases the distance to the goal. We terminate the episode when the quadruped flips over as no algorithm learned to reverse this (**Ant Maze Medium/Large**). We added a harder variant which requires the agent to push a block aside to reach one of the goals (**Ant Maze Push**).

5.1 Skill analysis

This section analyzes the skills learned by SPlaTES in isolation, to verify that they are predictable, diverse, and useful. For the sake of clarity, we first consider a toy environment with a velocity-controlled point mass. To probe the ability of the learned skills to compensate perturbations, we add temporally sparse noise of fixed magnitude and random direction to the velocity during training.

The left side of Figure 3 compares DADS to SPlaTES skills under a perturbation perpendicular to the predicted change in the state. While both methods learn a set of *diverse* skills, they differ in their robustness to noise: As SPlaTES is trained to produce *predictable transitions over K time steps*, its skills compensate the perturbation rapidly and stay close to the predicted trajectory. This reflex-like behavior is realized by the skill policy and does not require MPC. DADS, on the other hand, is oblivious of deviating from the predicted trajectory and maintains an offset. The right side of Figure 3 shows DADS and SPlaTES controlling a quadruped. At one time step, a large force is applied (marked by a red arrow). In this high-dimensional environment, the same behavior occurs: SPlaTES corrects the error resulting from the force while DADS cannot. We analyze this example in more detail in Section A.2.

In principle, MPC can help correct errors but in real-world applications it is often infeasible to run it at every time step due to computational constraints. It is therefore desirable to distill error-correcting behavior into a fast skill policy.

Figure 4 shows trajectories predicted by the abstract SPlaTES world model based on brown-noise sequences of skill vectors. Although the latent state space is learned only from the reward signal, the skills focus on manipulating the block or moving the quadruped. They are thus *useful* for the task. The temporal extent of the skills moreover facilitates exploration. The sampled skill sequences consequently cover most of the work space and maze.

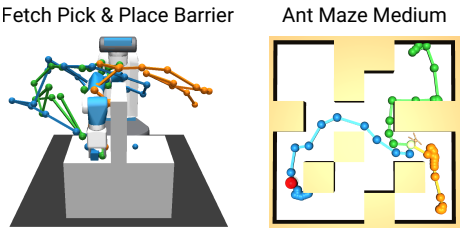


Figure 4: SPlaTES predictions for random brown-noise skill sequences. Most of the work space of the robot and the quadruped maze are covered.

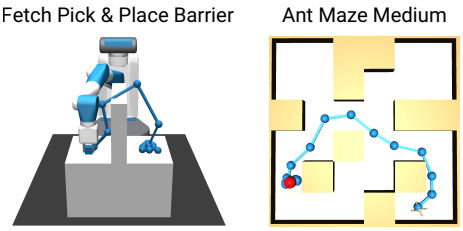


Figure 5: SPlaTES plans on the Fetch P&P Barrier and Ant Maze Medium tasks. Note that the skill sequence extends over the whole episode.

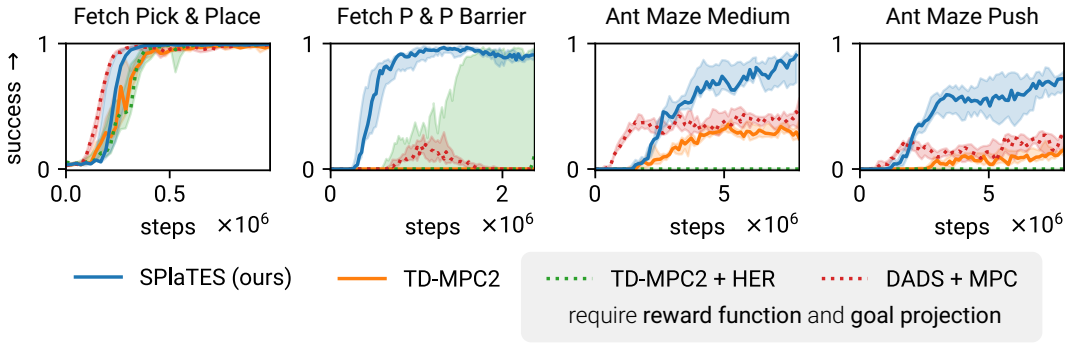


Figure 6: Success rates over the course of training. The shaded areas indicate the region between the 20% and 80% percentiles across five seeds. The lines correspond to the median. Methods using privileged information such as the reward function and the goal projection are shown as dotted lines.

5.2 Comparative analysis

Does planning over predictable skills enable model-based RL to solve long-horizon continuous control tasks? To answer this question, we compare SPlaTES to a set of competitive baselines on increasingly challenging variants of two RL domains (de Lazcano et al., 2023). All learning is done online and from scratch. We use a dense reward proportional to the negative Euclidean distance to the goal, as our focus is on task-related behavior rather than intrinsic motivation. All tasks are continuing: the environment does not terminate when the goal is reached but truncates after a time limit. On the higher level, SPlaTES plans over the whole episode, corresponding to an effective horizon of up to 1400 environment steps.

Figure 6 shows how the success rates of the baselines and SPlaTES evolve during training. Note that HER and DADS require access to the reward function and goal projection while SPlaTES does not. All methods solve the basic *Fetch Pick and Place* task with SPlaTES being competitive in terms of sample efficiency. On the more challenging *Fetch Pick and Place Barrier* task SPlaTES quickly discovers how to lift the block over the barrier. In contrast to this, TD-MPC2 remains in the local optimum of pressing the block against the base of the barrier. Despite having a two-times larger MPC horizon and replanning every time step, DADS only inconsistently finds the path across the barrier later in the training. HER only succeeds in less than half of the seeds in finding a trajectory around the barrier. Note that we ran HER with a dense and a sparse reward and report results for whichever worked best on each environment.

On the two Ant Maze tasks TD-MPC2 and DADS can solve combinations of initial position and goal that can be connected by greedily following the gradient of the reward function while sliding along walls. Only SPlaTES succeeds in more challenging episodes that require going around an obstacle for an extended period of time before turning back towards the goal again. In principle, increasing the MPC horizon in TD-MPC2 and DADS should enhance performance but in practice this is infeasible because (i) rolling the model out for hundreds of time steps becomes prohibitively expensive, and (ii) the quality of the predictions drastically deteriorates. We found HER not to work in this setting. We hypothesize that the lack of success could be caused by a complete lack of overlap

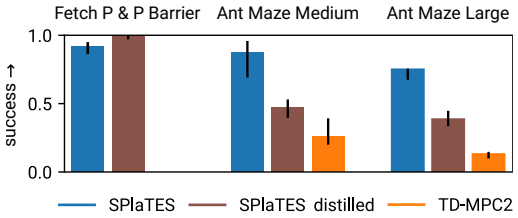


Figure 7: Distillation of the hierarchical SPlaTES agent into a flat TD-MPC2 model. The success rate increases on the Fetch Pick & Place Barrier task, whereas it drops significantly in the more challenging Ant Maze environments. Median of the final performance across five seeds with error bars corresponding to the 20% and 80% percentiles.

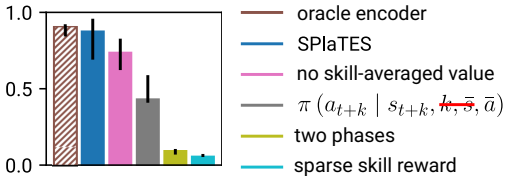


Figure 8: Ablations of SPlaTES on Ant Maze Medium. Oracle encoder: A handcrafted encoder; No skill-averaged value: Bootstrap directly from Q function at end of skill; No k and $\bar{s}$: No conditioning on intra-skill step and start state; Two phases: Learn skills first, then plan; Sparse skill reward: Give Equation 3 as reward at end of skill. Median across five seeds with error bars corresponding to the 20% and 80% percentiles.

between environment and hindsight goals, and a lack of exploration due to the danger of flipping over and terminating. Hence, only SPlaTES performs well on these challenging long-horizon tasks. We additionally compare to a competitive model-free hierarchical baseline (Levy et al., 2019) in Section A.3, and find that it learns considerably slower than SPlaTES.

The superior asymptotic performance of SPlaTES on the more challenging tasks can be attributed to two factors: improved exploration and better long-term credit assignment. Figure 4 illustrates how temporally extended skills in combination with brown noise in the iCEM sampling process aid exploration. To test whether SPlaTES improves credit assignment, we distill the hierarchical agent into a flat TD-MPC2 model. To this end, we begin by training SPlaTES until the performance plateaus. We then start to train a TD-MPC2 agent in parallel which has access to the replay buffer of the SPlaTES skills. We additionally switch randomly (within episodes) between the two agents to make sure new experience is collected in the whole maze. Figure 7 shows that this process leads to a new agent with close-to-optimal performance on Fetch Pick & Place Barrier. However, on the long-horizon tasks Ant Maze Medium and Ant Maze Large, the performance of the distilled agent is significantly lower than that of SPlaTES despite having access to successful trajectories. For the medium-sized maze the decrease in the success rate can be attributed to failing to reach the goal exactly. On Ant Maze Large, on the other hand, myopic behavior reappears in the distilled agent (we show an example case in Section A.1). We thus conclude that the long-term credit assignment achieved by SPlaTES via abstract planning is difficult to reproduce with flat TD learning.

5.3 Ablative analysis

To gauge the impact of different components of SPlaTES on its performance, we ablate them individually on Ant Maze Medium in Figure 8. Replacing the learned encoder with a hand-crafted one (*encoder oracle*) brings only a very modest performance benefit. Using the Q-function conditioned on the next skill vector directly in the skill-learning TD-target instead of a learned approximation of the expectation over all skill values results in a noisier target, and a decrease in performance of 15%. Ablating the conditioning of the skill policy (on the intra-skill time step and the abstract state the skill started in) removes the ability to correct errors in the skill trajectory. This results in a performance drop of about 45%. Dividing training into a skill learning phase with random skill vectors and a planning phase with frozen skills results in a low success rate. Hence, guiding the skills with high-level planning is useful for learning relevant skills. Finally, providing the approximation of the mutual information $\phi\left(f(s_{t+K})\right)$ (see Equation 3) as a sparse reward at the end of each skill execution in place of the dense increments $\phi\left(f(s_{t+k+1})\right) - \phi\left(f(s_{t+k})\right)$ has the greatest impact on performance. Skill learning slows dramatically, resulting in a success rate of around 5%.

6 Related work

Model-based RL uses learned world models (Schmidhuber, 1990) to predict state transitions and rewards. This enables differentiating through the model (Deisenroth & Rasmussen, 2011), planning

(Hafner et al., 2019b), or generating synthetic experience for model-free RL algorithms (Sutton, 1991; Hafner et al., 2019a). Recently, the latter two approaches have been combined in hybrid methods that plan over a small number of time steps while accounting for long-term effects with a learned value function (Schrittwieser et al., 2020; Hansen et al., 2022). The problem of compounding model errors (Lambert et al., 2022) has been addressed in several ways: Increasing one-step model accuracy by improving data collection or architecture choices can increase performance (Plaat et al., 2023), but does not address all of the challenges discussed in Section 3. Branching off short rollouts from observed states (Janner et al., 2019) avoids compounding model errors but neglects long-term credit assignment. Directly predicting states multiple time steps in the future can increase accuracy in some environments (Neitz et al., 2018; Asadi et al., 2019), but results are generally highly dependent on the data-collection policy (Lambert et al., 2021). Predicting entire trajectories from offline data is a promising research direction but also entangles policy and environment dynamics (Janner et al., 2021; Ding et al., 2024). Finally, keeping track of epistemic and aleatoric uncertainty (Chua et al., 2018) can quantify the problem but does not directly enable longer rollouts.

Hierarchical RL (HRL) (Hutsebaut-Buysse et al., 2022) splits up decision making into multiple interconnected levels of abstraction: A higher level chooses temporally extended courses of actions and a lower level executes them (Dayan & Hinton, 1992). Thus, the problem horizon is reduced by temporal abstraction, facilitating credit assignment and exploration. The options framework (Sutton et al., 1999; Barto & Mahadevan, 2003; Bacon et al., 2017) formalizes the notion of closed-loop courses of action (also referred to as skills). Many HRL frameworks break long-horizon tasks down into a sequence of subgoals (Nachum et al., 2018), enabling sample-efficient hindsight relabeling techniques (Andrychowicz et al., 2017; Levy et al., 2019). However, the projection to the subgoal space is usually designed manually, as learning it is challenging (Nachum et al., 2019; Choi et al., 2021). Picking a subgoal furthermore requires checking whether it is feasible in the given situation (Zhang et al., 2023). In contrast to this, the skills we learn are always applicable, which simplifies planning. Hansen et al. (2022) learn partial option models, predicting the outcome of options when available, but use a fixed set of handcrafted options. Hafner et al. (2022) learn a latent goal space, but do not use an abstract model for planning. Park et al. (2023) use an intermediate representation of an offline-learned value function as goal space, which is, however, not directly applicable in the online case when no high-quality value function is available yet. Shi et al. (2023) learn skills together with a model of skill outcomes offline, and solve downstream tasks with MPC. The success of this approach hinges on the quality of the pre-collected dataset, and does not explicitly encourage predictability of skill coutcomes. Xie et al. (2021) learn skills with a sum of intrinsic and extrinsic reward but plan with a learned 'flat' dynamics model that predicts atomic transitions based on primitive actions, i.e., without abstraction in terms of actions and time.

Skill discovery aims to learn useful behaviors that can be combined to solve downstream tasks. Gregor et al. (2016) propose Variational Intrinsic Control (VIC), which maximizes the mutual information (MI) between a skill and the state it terminates in conditioned on the start state. The main differences to our skill-learning objective are: (i) We use a dense reward (Equation 4), (ii) we use a forward model instead of a discriminator to approximate MI (Sharma et al., 2020b), (iii) we condition the skill policy on the intra-skill time step k and the abstract state it started in as discussed in Section 4.3, and (iv) we consider a learned abstract state. Gregor et al. (2016) furthermore argue that training VIC becomes unstable when combined with function approximation. However, we found that our reward combined with appropriate learning rates for the model and skill policy results in stable training for SPlaTES. Eysenbach et al. (2019) maximize the MI between the skill vector and the next state, approximating it with a discriminator. This results in diverse skills that seek out different parts of the state space but are not necessarily predictable on shorter time scales. Sharma et al. (2020b), on the other hand, focus on controlling atomic transitions by additionally conditioning on the current state. We propose to strike a balance by controlling abstract, temporally extended transitions. Achterhold et al. (2023) learn skills using a given discrete state abstraction to play physically-embedded board games. The discrete skills are trained with the sparse VIC reward, and correspond to actions in a symbolic forward model, which is then used for high-level planning. Various unsupervised skill learning methods use inductive biases to discover meaningful skills when neither a compact latent space nor a task reward are given (Park et al., 2022, 2024; Machado et al., 2018).

7 Conclusion

In this work, we introduced SPlaTES, a sample-efficient hierarchical RL algorithm that learns temporally extended skills on the lower level, and an abstract world model that predicts skill outcomes on the higher level. We have demonstrated that our skill learning objective results in (i) diverse, predictable, and task-related behavior, and (ii) the ability to counteract errors, which improves the reliability of long model rollouts. By performing MPC on different timescales on both levels of the model-based hierarchy, we outperform competitive model-based, skill-based and hierarchical baselines on challenging long-horizon tasks. Distilling the hierarchical agent into a flat TD-MPC2 model resulted in the reoccurrence of myopic behavior, indicating that our model-based hierarchy performs credit assignment at time scales that are difficult to achieve with non-hierarchical TD learning.

Limitations: While learning the encoder removes the requirement to design it manually, we still need to choose an appropriate dimension for the abstract state. Moreover, using gradients from the high-level reward loss for encoder learning requires sufficiently dense rewards. Although taking a high-level value function into account could lift this requirement, we found learning such a value function challenging, particularly in the early phase of training. More generally, temporal abstraction results in a scarcity of training data on the higher level. Therefore, more sample-efficient supervised learning methods or appropriate data augmentation techniques are needed to further improve sample efficiency.

Acknowledgments

We thank Marco Bagatella for valuable feedback and insightful comments on a draft of this manuscript. Nico Gürtler is funded by the European Union (ERC, REAL-RL, 101045454). Georg Martius is a member of the Machine Learning Cluster of Excellence, EXC number 2064/1 – Project number 390727645. Finally, this work was supported by the German Federal Ministry of Education and Research (BMBF): Tübingen AI Center, FKZ: 01IS18039A.

References

Jan Achterhold, Markus Krimmel, and Joerg Stueckler. Learning temporally extended skills in continuous domains as symbolic actions for planning. In *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pp. 225–236. PMLR, 14–18 Dec 2023.

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Kavosh Asadi, Dipendra Misra, Seungchan Kim, and Michel L Littman. Combating the compounding-error problem with a multi-step model. *arXiv preprint arXiv:1905.13320*, 2019.

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.

Marco Bagatella and Georg Martius. Goal-conditioned offline planning from curious exploration. In *Advances in Neural Information Processing Systems*, volume 36, pp. 15358–15383. Curran Associates, Inc., 2023.

Andrew G Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete event dynamic systems*, 13:341–379, 2003.

Jongwook Choi, Archit Sharma, Honglak Lee, Sergey Levine, and Shixiang Shane Gu. Variational empowerment as representation learning for goal-based reinforcement learning. *arXiv preprint arXiv:2106.01404*, 2021.

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

Peter Dayan and Geoffrey E Hinton. Feudal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 5. Morgan-Kaufmann, 1992.

Rodrigo de Lazcano, Kallinteris Andreas, Jun Jet Tai, Seungjae Ryan Lee, and Jordan Terry. Gymnasium robotics, 2023.

Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pp. 465–472, 2011.

Zihan Ding, Amy Zhang, Yuandong Tian, and Qinqing Zheng. Diffusion world model: Future modeling beyond step-by-step rollout for offline reinforcement learning. *arXiv preprint arXiv:2402.03570*, 2024.

Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. In *International Conference on Learning Representations*, 2019.

Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

Karol Gregor, Danilo Jimenez Rezende, and Daan Wierstra. Variational intrinsic control. *arXiv preprint arXiv:1611.07507*, 2016.

Nico Gürtler, Dieter Büchler, and Georg Martius. Hierarchical reinforcement learning with timed subgoals. In *Advances in Neural Information Processing Systems*, volume 34, pp. 21732–21743. Curran Associates, Inc., 2021.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019a.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 2555–2565. PMLR, 09–15 Jun 2019b.

Danijar Hafner, Timothy P Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. In *International Conference on Learning Representations*, 2021.

Danijar Hafner, Kuang-Huei Lee, Ian Fischer, and Pieter Abbeel. Deep hierarchical planning from pixels. In *Advances in Neural Information Processing Systems*, volume 35, pp. 26091–26104. Curran Associates, Inc., 2022.

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

Nicklas Hansen, Hao Su, and Xiaolong Wang. TD-MPC2: Scalable, robust world models for continuous control. In *The Twelfth International Conference on Learning Representations*, 2024.

Nicklas A Hansen, Hao Su, and Xiaolong Wang. Temporal difference learning for model predictive control. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 8387–8406. PMLR, 17–23 Jul 2022.

Matthias Hutsebaut-Buysse, Kevin Mets, and Steven Latré. Hierarchical reinforcement learning: A survey and open research challenges. *Machine Learning and Knowledge Extraction*, 4(1):172–221, 2022. ISSN 2504-4990. DOI: 10.3390/make4010009.

Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Michael Janner, Qiyang Li, and Sergey Levine. Offline reinforcement learning as one big sequence modeling problem. In *Advances in Neural Information Processing Systems*, volume 34, pp. 1273–1286. Curran Associates, Inc., 2021.

Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1):99–134, 1998.

Nathan Lambert, Albert Wilcox, Howard Zhang, Kristofer S. J. Pister, and Roberto Calandra. Learning accurate long-term dynamics for model-based reinforcement learning. In *2021 60th IEEE Conference on Decision and Control (CDC)*, pp. 2880–2887, 2021. DOI: 10.1109/CDC45484.2021.9683134.

Nathan Lambert, Kristofer Pister, and Roberto Calandra. Investigating compounding prediction errors in learned dynamics models. *arXiv preprint arXiv:2203.09637*, 2022.

Andrew Levy, Robert Platt, and Kate Saenko. Learning multi-level hierarchies with hindsight. In *International Conference on Learning Representations*, 2019.

Marlos C. Machado, Clemens Rosenbaum, Xiaoxiao Guo, Miao Liu, Gerald Tesauro, and Murray Campbell. Eigenoption discovery through the deep successor representation. In *International Conference on Learning Representations*, 2018.

Ofir Nachum, Shixiang (Shane) Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. Near-optimal representation learning for hierarchical reinforcement learning. In *International Conference on Learning Representations*, 2019.

Alexander Neitz, Giambattista Parascandolo, Stefan Bauer, and Bernhard Schölkopf. Adaptive skip intervals: Temporal abstraction for recurrent dynamical models. *Advances in Neural Information Processing Systems*, 31, 2018.

Edward Ott. *Chaos in dynamical systems*. Cambridge university press, 2002.

Fabio Pardo, Arash Tavakoli, Vitaly Levdik, and Petar Kormushev. Time limits in reinforcement learning. In *International Conference on Machine Learning*, pp. 4045–4054. PMLR, 2018.

Seohong Park, Jongwook Choi, Jaekyeom Kim, Honglak Lee, and Gunhee Kim. Lipschitz-constrained unsupervised skill discovery. In *International Conference on Learning Representations*, 2022.

Seohong Park, Dibya Ghosh, Benjamin Eysenbach, and Sergey Levine. Hiql: Offline goal-conditioned rl with latent states as actions. In *Advances in Neural Information Processing Systems*, volume 36, pp. 34866–34891. Curran Associates, Inc., 2023.

Seohong Park, Oleh Rybkin, and Sergey Levine. METRA: Scalable unsupervised RL with metric-aware abstraction. In *The Twelfth International Conference on Learning Representations*, 2024.

Cristina Pinneri, Shambhuraj Sawant, Sebastian Blaes, Jan Achterhold, Joerg Stueckler, Michal Rolinek, and Georg Martius. Sample-efficient cross-entropy method for real-time planning. In *Proceedings of the 2020 Conference on Robot Learning*, volume 155 of *Proceedings of Machine Learning Research*, pp. 1049–1065. PMLR, 16–18 Nov 2021.

Aske Plaat, Walter Kosters, and Mike Preuss. High-accuracy model-based reinforcement learning, a survey. *Artificial Intelligence Review*, 56(9):9541–9573, 2023.

Matthias Plappert, Marcin Andrychowicz, Alex Ray, Bob McGrew, Bowen Baker, Glenn Powell, Jonas Schneider, Josh Tobin, Maciek Chociej, Peter Welinder, Vikash Kumar, and Wojciech Zaremba. Multi-goal reinforcement learning: Challenging robotics environments and request for research, 2018.

Jürgen Schmidhuber. *Making the world differentiable: on using self supervised fully recurrent neural networks for dynamic reinforcement learning and planning in non-stationary environments*, volume 126. Inst. für Informatik, 1990.

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

Archit Sharma, Michael Ahn, Sergey Levine, Vikash Kumar, Karol Hausman, and Shixiang Gu. Emergent real-world robotic skills via unsupervised off-policy reinforcement learning. *arXiv preprint arXiv:2004.12974*, 2020a.

Archit Sharma, Shixiang Gu, Sergey Levine, Vikash Kumar, and Karol Hausman. Dynamics-aware unsupervised discovery of skills. In *International Conference on Learning Representations*, 2020b.

Lucy Xiaoyang Shi, Joseph J Lim, and Youngwoon Lee. Skill-based model-based reinforcement learning. In *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pp. 2262–2272. PMLR, 14–18 Dec 2023.

Jean-Jacques E Slotine, Weiping Li, et al. *Applied nonlinear control*, volume 199. Prentice hall Englewood Cliffs, NJ, 1991.

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

Richard S Sutton, Andrew G Barto, and Co-Director Autonomous Learning Laboratory Andrew G Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

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

Kevin Xie, Homanga Bharadhwaj, Danijar Hafner, Animesh Garg, and Florian Shkurti. Latent skill planning for exploration and transfer. In *International Conference on Learning Representations*, 2021.

Tianren Zhang, Shangqi Guo, Tian Tan, Xiaolin Hu, and Feng Chen. Adjacency constraint for efficient hierarchical reinforcement learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(4):4152–4166, 2023. DOI: 10.1109/TPAMI.2022.3192418.

A Additional results

In this section, we provide additional results and visualizations, comparing SPlaTES to baselines and analyzing its skill training.

A.1 Distillation of a SPlaTES agent into a flat TD-MPC2 model

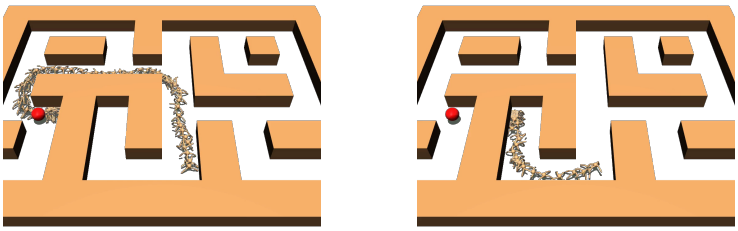
As discussed in Section 5.2, distilling a SPlaTES agent into a flat TD-MPC2 agent on Ant Maze Large resulted in a significant drop in the success rate for two reasons: (i) The agent often does not match the goal position precisely enough, and (ii) it regresses to myopic behavior that leads it into local minima instead of going around obstacles. Note that the task is continuing and lasts for a 1000 time steps. We used a discount factor of 0.995 as we did not see any improvements in training from scratch or distillation when increasing it in grid searches. Figure 9 shows an example of issue (ii), i.e., myopic behavior reappearing even though the hierarchical agent generates close to optimal data.

A.2 Execution of a fixed skill sequence with a perturbation

In Section 5.1, qualitative results for an Ant quadruped executing a fixed skill sequence while being pushed by a large force in one time step have been shown to illustrate the compensation of perturbations by SPlaTES. In Figure 10, ten rollouts generated by DADS and SPlaTES in this scenario are shown and analyzed. We conclude that error-correcting behavior occurs consistently when using SPlaTES.

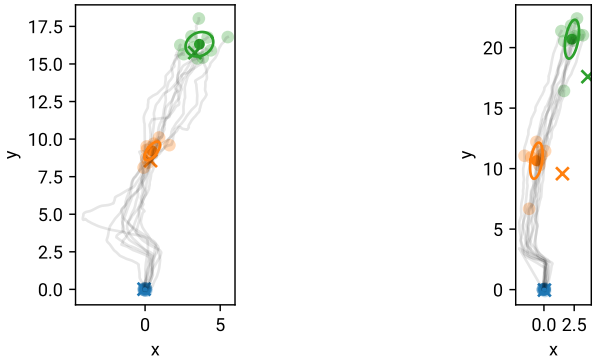
A.3 Comparison to a hierarchical baseline

In this section, we compare SPlaTES to a hierarchical baseline. **Hierarchical Actor-Critic (HAC)** (Levy et al., 2019) is a model-free hierarchical RL algorithm that breaks a task down into a series of subgoals. The high-level policy chooses the next subgoal whereas the low-level policy pursues it. Hindsight relabeling is used to improve sample efficiency. HAC requires privileged information



(a) SPlaTES (b) Distilled TD-MPC2 agent

Figure 9: Failure to distill long-horizon behavior into flat TD-MPC2 agent: Even though the distilled TD-MPC2 agent has access to clos-to-optimal data generated by the hierarchical agent, it regresses to suboptimal, myopic behavior.



(a) SPlaTES (b) DADS

Figure 10: Execution of a fixed skill sequence with the Ant quadruped while experiencing a force to the negative x direction in a single time step. Trajectories are rendered as black, translucent lines, and x-y coordinates at multiple of the control interval K of SPlaTES are shown as colored, translucent circles. At multiples of K time steps, a Gaussian is fitted to the x-y coordinates, and depicted as colored circles. The x-y coordinate predicted by the world model is shown as a cross at these time steps. Note how SPlaTES compensates the 'kick', while DADS maintains the offset in its trajectory which is caused by it.

in the form of a map to the subgoal space and the reward function. We use the gym-compatible implementation of HAC from Gürtler et al. (2021).

We did not succeed in getting HAC to learn on the environments considered in the main text. We hypothesize that there are several factors contributing to this failure to learn: (i) The Fetch variants have sparse interactions with the object which is known to cause issues, in particular with methods that rely on hindsight relabeling (as a vast majority of the hindsight goals and actions correspond to the cube being stationary). (ii) The considered environments (except for Fetch Pick & Place) have almost no overlap between hindsight goals and environment goals in the initial phase of training. (iii) HAC terminates the low-level episode after each skill and is thus not learning to chain skills. In the Ant environments this results in flipping over at the end of skills.

To be able to compare to HAC, we modified Ant Maze Medium by making the actuators weaker by a factor of 10. This was also done in Levy et al. (2019), probably to avoid issue (iii). This prevented the Ant from flipping over. To also circumvent issue (ii), we give the higher level access to the dense reward function. These changes resulted in HAC learning, albeit slowly. Figure 11 shows the success rate and return of SPlaTES and HAC on this modified environment (as the success rate of HAC stays at zero). SPlaTES outperforms HAC in terms of sample efficiency, probably due to (i) TD-MPC2's sample efficiency on the lower level, (ii) the dense skill learning reward, and (iii) better targeted

exploration due to high-level planning with the abstract world model. We additionally observed that the higher level of HAC only slowly moves the subgoals further away from the initial position of the Ant. This overly conservative behavior is probably caused by the penalty the higher level receives when proposing an infeasible subgoal. As a result, exploration is slowed down significantly. In contrast to this, all skills are feasible and SPlaTES therefore does not need such a penalty.

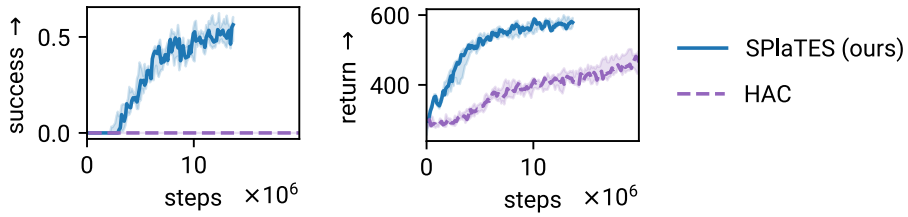


Figure 11: Comparison of SPlaTES to HAC on a modified version of Ant Maze Medium with a 'weaker' (and therefore slower) Ant: SPlaTES outperforms HAC in terms of learning speed. The shaded areas indicate the region between the 20% and 80% percentiles across five seeds. The lines correspond to the median. HAC uses privileged information in the form of the goal projection.

A.4 Decoder

To analyze what aspects of the state the learned encoder extracts, we trained an MLP decoder independently of training SPlaTES (no gradients were allowed to flow back). Figure 12 shows the normalized reconstruction error (normalized root mean square error) of different components of the states and relative offset to the desired goal and norm of this offset. We conclude that the encoder focuses on the achieved and desired goal as they are crucial for fitting the reward.

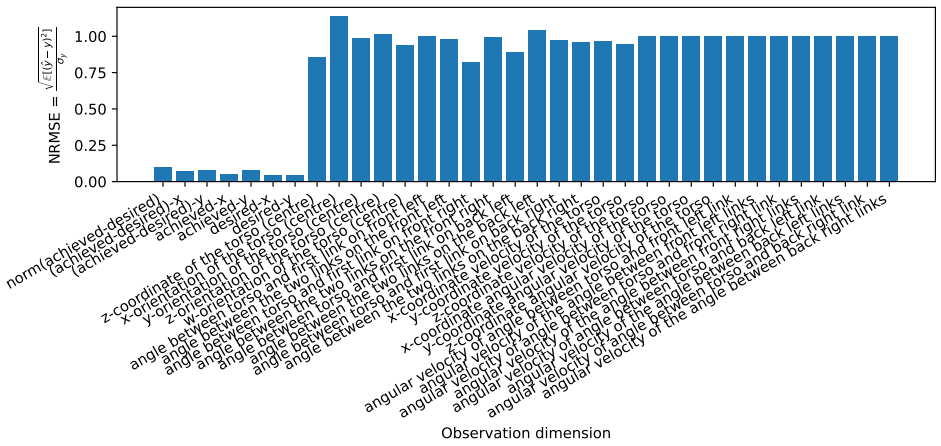


Figure 12: Reconstruction error of observation dimensions from latent state and context(normalized root mean square error): A decoder is trained (without propagating any gradients to the model) to reconstruct the observation from the latent state and context.

A.5 Visualizations

We provide additional visualizations of the learned skills in this section.

B Algorithm

We give additional details on the algorithm in this section, in particular the derivation of the skill learning reward and implementation details.

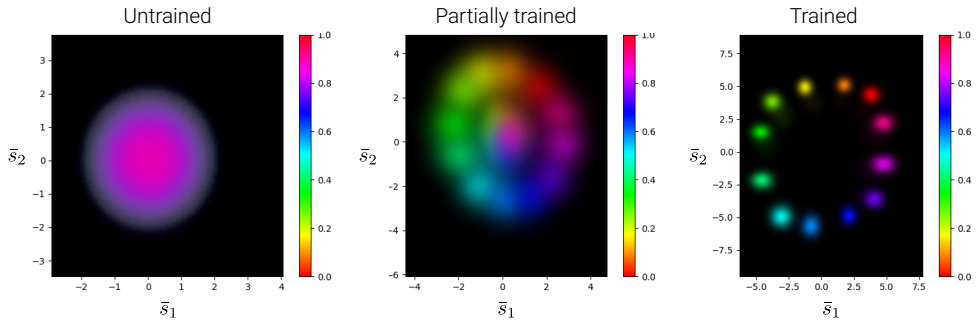


Figure 13: Predicted distributions of the abstract state delta after a skill execution at different stages of training (Ant Maze Environment): Twelve skills are sampled uniformly on the unit circle in skill vector space and color coded. The next state distributions predicted by the abstract world model are visualized by color, with transparency determined by density. Note how at the intermediate training stage some probability mass is at the origin for all skills. This corresponds to the skill being unable to move, for example due to being off the ground or stuck on an edge. In the final model (right), this probability mass mostly disappeared as the skills have become competent at locomotion.

B.1 From mutual information to learning temporally extended skills

This section describes in detail how we approximate the mutual information in Equation 2, and how we use it to define a dense skill learning reward.

B.2 Approximating the mutual information

To obtain our skill learning reward, we first have to approximate the mutual information of the next abstract state and the skill vector, conditioned on the current skill vector,

$$I(\bar{s}'; \bar{a} \mid \bar{s}) = H(\bar{s}' \mid \bar{s}) - H(\bar{s}' \mid \bar{s}, \bar{a}) \tag{5}$$

$$= \int\int p\left(\bar{s}, \bar{a}, \bar{s}'\right) \log \frac{p\left(\bar{s}' \mid \bar{s}, \bar{a}\right)}{p\left(\bar{s}' \mid \bar{s}\right)} \, d\bar{a} \, d\bar{s}' \, d\bar{s} \tag{6}$$

$$= \mathbb{E}_{\bar{s},\bar{a},\bar{s}'} \left[\log \frac{\bar{p}\left(\bar{s}' \mid \bar{s}, \bar{a}\right)}{p\left(\bar{s}' \mid \bar{s}\right)} \right] . \tag{7}$$

The joint distribution of abstract state, skill vector, and next state reads

$$p\left(\bar{s}, \bar{a}, \bar{s}'\right) = p\left(\bar{s}\right) p\left(\bar{a} \mid \bar{s}\right) \bar{p}\left(\bar{s}' \mid \bar{s}, \bar{a}\right) . \tag{8}$$

We would like the skills to fill the whole skill vector space uniformly, i.e., we want to maximize the diversity of all available skills, and not only those chosen by the planner. We therefore choose a uniform distribution for the skill vector, $\bar{a} \sim \mathcal{U}(\bar{A})$ and sample independently from the abstract state $\bar{s}$. In practice $p\left(\bar{a} \mid \bar{s}\right)$ is determined by the high-level planning, interleaved with randomly sampled skills for exploration. This will be accounted for by importance sampling in the next subsection. The joint distribution therefore simplifies to

$$p\left(\bar{s}, \bar{a}, \bar{s}'\right) = p\left(\bar{s}\right) p\left(\bar{a}\right) \bar{p}\left(\bar{s}' \mid \bar{s}, \bar{a}\right) . \tag{9}$$

We now follow Sharma et al. (2020b) for the rest of the derivation. We first obtain a variational lower bound for the mutual information by replacing the true dynamics $\bar{p}$ of the abstract POMDP with the approximation $\hat{\bar{p}}$ our world model learned,

$$I(\bar{s}'; \bar{a} \mid \bar{s}) = \mathbb{E}_{\bar{s},\bar{a},\bar{s}'} \left[\log \frac{\bar{p}\left(\bar{s}' \mid \bar{s}, \bar{a}\right)}{p\left(\bar{s}' \mid \bar{s}\right)} \right] \tag{10}$$

$$= \mathbb{E}_{\bar{s},\bar{a},\bar{s}'} \left[\log \frac{\hat{\bar{p}}\left(\bar{s}' \mid \bar{s}, \bar{a}\right)}{p\left(\bar{s}' \mid \bar{s}\right)} \right] + \mathbb{E}_{\bar{s},\bar{a}} \left[\mathcal{D}_{\mathrm{KL}} \left(\bar{p}\left(\bar{s}' \mid \bar{s}, \bar{a}\right) \,||\, \hat{\bar{p}}\left(\bar{s}' \mid \bar{s}, \bar{a}\right) \right) \right] \tag{11}$$

$$\geq \mathbb{E}_{\bar{s},\bar{a},\bar{s}'} \left[\log \frac{\hat{\bar{p}}\left(\bar{s}' \mid \bar{s}, \bar{a}\right)}{p\left(\bar{s}' \mid \bar{s}\right)} \right] , \tag{12}$$

where we used the non-negativity of the Kullback-Leibler divergence.

Maximizing the variational lower bound involves minimizing the Kullback-Leibler divergence. We realize this by maximizing the log likelihood of the abstract transitions when training the abstract world model.

We approximate the marginal distribution $p\left(\bar{s}' \mid \bar{s}\right)$ with Monte Carlo sampling. To this end, we sample N skill vectors $\bar{a}_i \sim \mathcal{U}(\bar{A})$, replace the integration with an average, and approximate the dynamics of the POMDP with the world model again:

$$p\left(\bar{s}' \mid \bar{s}\right) = \int p(\bar{a}) \bar{p}\left(\bar{s}' \mid \bar{s}, \bar{a}\right) \, d\bar{a} \tag{13}$$

$$\approx \frac{1}{N} \sum_{i=1}^{N} \bar{p}\left(\bar{s}' \mid \bar{s}, \bar{a}_i\right) \tag{14}$$

$$\approx \frac{1}{N} \sum_{i=1}^{N} \hat{\bar{p}}\left(\bar{s}' \mid \bar{s}, \bar{a}_i\right) \tag{15}$$

$$\tag{16}$$

This yields the following approximation for the mutual information:

$$I(\bar{s}'; \bar{a} \mid \bar{s}) \approx \mathbb{E}_{\bar{s}, \bar{a}, \bar{s}'} \left[\log \frac{\hat{\bar{p}}\left(\bar{s}' \mid \bar{s}, \bar{a}\right)}{\frac{1}{N} \sum_{i=1}^{N} \hat{\bar{p}}\left(\bar{s}' \mid \bar{s}, \bar{a}_i\right)} \right] \tag{17}$$

B.3 From mutual information maximization to an RL reward

We now consider skill learning via RL. One skill learning RL episode corresponds to the execution of one skill for K time steps, starting in the abstract state $\bar{s} = f(s_t)$ and ending in $\bar{s}' = f(s_{t+K})$. The skill policy is conditioned on $\bar{a}$. The distribution $p\left(\bar{a}, \bar{s}\right)$ therefore plays a similar role as a distribution of goals or tasks in multitask RL.

We can now tentatively identify the expected return of an RL episode with the approximated mutual information

$$\mathbb{E}_{\bar{s}, \bar{a}}\left[G\right] = \mathbb{E}_{\bar{s}, \bar{a}} \left[\rho\left(\bar{s}, \bar{a}\right) \log \frac{\hat{\bar{p}}\left(\bar{s}' \mid \bar{s}, \bar{a}\right)}{\frac{1}{N} \sum_{i=1}^{N} \hat{\bar{p}}\left(\bar{s}' \mid \bar{s}, \bar{a}_i\right)} \right] . \tag{18}$$

Define the potential

$$\phi(\bar{s}'; \bar{s}, \bar{a}) \coloneqq \log \frac{\hat{\bar{p}}\left(\bar{s}' \mid \bar{s}, \bar{a}\right)}{\frac{1}{N} \sum_{i=1}^{N} \hat{\bar{p}}\left(\bar{s}' \mid \bar{s}, \bar{a}_i\right)} , \tag{19}$$

and the dense reward

$$r_{\text{skill}}(s_{t+k}, a, s_{t+k+1}; \bar{s}) \coloneqq \phi(f(s_{t+k+1})); \bar{s}, \bar{a}) - \phi(f(s_{t+k}); \bar{s}, \bar{a}) . \tag{20}$$

Then the return becomes a telescoping sum and the expected return is equal to the approximation to the mutual information (equation 17) up to a constant term (as $\bar{s}$ and $\bar{a}$ are fixed during a skill learning RL episode) which does not influence the optimal skill policy,

$$\mathbb{E}_{\bar{s}, \bar{a}}\left[G\right] = \mathbb{E}_{\bar{s}, \bar{a}}\left[\phi(\bar{s}'; \bar{s}, \bar{a}) - \phi(\bar{s}; \bar{s}, \bar{a})\right] . \tag{21}$$

Hence, applying any suitable RL algorithm to the reward defined in Equation 20, maximizes an approximation to the mutual information $I(\bar{s}'; \bar{a} \mid \bar{s})$.

B.4 Planning without the Markov property

As discussed in Section 4.1, SPlaTES performs MPC using abstract states $\bar{s} \in \bar{\mathcal{S}}$ which, in general, lack the Markov property. In principle, the information missing from $\bar{s}$ could introduce a large amount of uncertainty into the abstract world model $\hat{\bar{\mathcal{M}}}$. So why are the plans that SPlaTES finds useful?

The answer to this question lies in the joint training of the abstract world model and the skill policy. When the higher level picks a skill vector, it does not know the details of the environment state.

However, *the skill policy does see the full environment state*, and is trained to reliably bring about the abstract state predicted by $\hat{\bar{\mathcal{M}}}$. More concretely, the skill-learning objective involves minimizing the entropy $H\left(\bar{s}_{t+K} \mid \bar{s}_t, \bar{a}_t\right)$ (see Equation 1), as approximated by the term $\log \hat{\bar{p}}\left(\bar{s}_{t+K} \mid \bar{s}_t, \bar{a}_t\right)$ in the reward (see Equation 3). This means that the skill policy will actively try to keep the abstract transition dynamics $p\left(\bar{s}_{t+K} \mid \bar{s}_t, \bar{a}_t\right)$ as predictable as possible, regardless of the history of the episode. As a side effect, the the dynamics will be approximately Markovian, i.e, loosely speaking,

$$p\left(\bar{s}_{t+K} \mid \bar{s}_t, \bar{a}_t\right) \approx p\left(\bar{s}_{t+K} \mid \bar{s}_t, \bar{a}_t, \ldots, \bar{s}_1, \bar{a}_1\right), \ \forall \bar{s}_{t-1}, \bar{a}_{t-1}, \ldots, \bar{s}_1, \bar{a}_1 \ . \tag{22}$$

How well the Markov property can be achieved depends on several factors:

- The longer the duration of the skill execution in relation to the intrinsic time scale of the environment, the more time is available for the skill policy to bring about the predicted abstract state. Consider a quadruped navigation task as an example. The abstract state may focus on the position but lack information about the velocity. In this case, the skill execution should be long enough to overcome the inertia of the quadruped, and achieve a suitable velocity to arrive at the predicted abstract state.
- The skill policy should try to stay flexible at the end of a skill execution by arriving in a state that not only corresponds to the predicted abstract state, but also serves as a good start state for the next skill (which is unknown before the higher level replans). For this reason, the skills are trained with chainability in mind as discussed in Section 4.3. To extend the quadruped example, high velocities or a loss of contact with the ground at the end of a skill would make it harder to execute the next skill and should therefore be avoided.

Approximately restoring the Markov property in this way comes at the price of conservatism. As the next skill vectors are not known in advance, the skill policy will try to be prepared for whichever one comes next which may slow it down (similar to jogging over the finish line versus 'diving' across it at the end of a race). The abstract world model picks this conservatism up, which further reinforces it. As a result, hierarchical SPlaTES policies are somewhat slower than an optimal flat policy would be. However, hierarchical RL is intended for situations where flat RL algorithms cannot find such an optimal flat policy in the first place. Furthermore, conditioning the skill policy not only on the current skill vector but also on the next (couple of) skill vectors could remedy this conservatism. As MPC provides such a sequence of skill vectors, this would be an interesting direction for future work.

B.5 Are the high-level dynamics stable?

The name of our method, Stable Planning with Temporally Extended Skills, contains the word *stable*, so in what sense is planning with our method stable?

Unfortunately, the maximization of the mutual information of the next abstract state and the skill vector (see Equation 1) in isolation implies only that the temporally abstract dynamics induced by the skills will be predictable. Since small changes in the initial abstract state and the skill vector could, in theory, have a large impact on the next abstract state, the resulting dynamics could potentially be unstable.

However, in practice the abstract dynamics are regularized by the neural network which is trained to fit them. In particular, multilayer perceptrons have a bias towards smooth functions, and we additionally initialize the dynamics as a linear function. Since the skills are in turn trained to achieve what this learned model predicts (see Equation 3), this bias directly influences the skills. The length scale over which the skills repel each other is furthermore kept fixed (see Section B.6) which 'locks' them in place, preventing the relation between skill vector and resulting abstract state from becoming unnecessarily nonlinear.

As a result, we empirically find that the high-level dynamics are indeed quite stable. This is verified by our analysis of the learned skills (see 5.1). In particular, the error-correcting behavior ensures that the abstract dynamics are much more well-behaved than the environment dynamics when compared over the same amount of environment time steps.

It is an interesting direction for future work to make this implicit bias towards smooth dynamics more explicit and controllable.

B.6 Implementation details

In this section, we provide details on the implementation of SPlaTES. For further details, we refer to the code and the configuration files (see project page).

Skill learning is implemented with a modified version of TD-MPC2 as it provides sample-efficient learning. Our modifications are (i) to keep track of the intra-skill time step k, and the abstract start state $\bar{s}$ of the skill, and the skill vector during rollouts (ii) to bootstrap from an additional learned Q-function (see below) at the end of a skill execution, (iii) to implement support for vectorized environments as we train with 12 environment instances for computational efficiency. We furthermore transform the state linearly before calculating the skill learning reward. This linear transformation is learned as the inverse of the covariance matrix of the abstract state deltas in the replay buffer. This ensures that length scales when calculating the skill learning reward are not arbitrary but correspond to typical changes brought about by the execution of a skill. We furthermore fix the standard deviation of the abstract world model when calculating the skill reward (similar to Sharma et al. (2020b)). This prevents premature convergence of the skills as it makes sure that the skills still repel each other in the abstract state space, even if they are already relatively precise. We use a skill duration K of 10 for the Fetch environments and 50 for the Ant Maze environments.

We learn a **expected Q-function** on the lower level to obtain a better target for bootstrapping at the end of a skill execution. The Q function is defined as

$$Q\left(a \mid s, k, \bar{s}, \right) \coloneqq \mathbb{E}_{\bar{a} \sim \mathcal{U}\left([-1,1]^{d_{\bar{\mathcal{A}}}}\right)}\left[Q\left(a \mid s, k, \bar{s}, z\right)\right] , \tag{23}$$

where $d_{\bar{\mathcal{A}}}$ denotes the dimension of the skill vector space. Intuitively, this Q-function learns the expected return-to-go averaged over all possible skills. By bootstrapping from it at the end of the skills, we ensure that all skill combinations are chainable.

An acceleration of skill learning in the initial phases of training can be achieved with a *symmetry breaking* phase. Initially, the randomly initialized skill policies do not manage to change the abstract state significantly. This makes the world model collapse to predicting very small, unstable skill deltas. As a result, the skill learning reward does not consistently encourage the skill to move into a specific direction in the abstract skill space. This issue can lead to a prolonged phase of 'collapsed' skills. To help the skills to differentiate, we initially calculate the skill learning reward with a random linear transition model. This breaks the symmetry and accelerates learning. We then switch to the learned model to learn the actual skill dynamics.

Improving exploration in skill learning is crucial for sample efficiency. We therefore clip the skill learning reward from below at zero for the initial phases of training on the Fetch tasks. This ensures that there is no penalty for exploring by moving the cube. We apply the same trick to the DADS baseline to ensure a fair comparison. On the Ant environments, we found this not to be necessary due to the absence of sparse contacts.

Encoder learning is implemented with a simple linear encoder as we found this to be sufficient for learning from states. It furthermore ensures that the encoder does not partly perform non-linear transformations needed for reward fitting. We split the output of the encoder up into a state $\bar{s}$ and a context $\bar{c}$, $f : \mathcal{S} \to \bar{\mathcal{S}} \times \bar{\mathcal{C}}$. The role of the context is to encode information about the task that is fixed in each episode. The context is only fed to the reward function but not to the learned abstract transition function. We found this distinction between state and context to not be strictly necessary but to improve generalization and the ability to visualize the abstract world model. We choose a dimension of 2 for the abstract state space for the Ant Maze environments, and 3 for the Fetch variants. This corresponds to the intrinsic spatial dimension.

The abstract world model consists of two learned components: The abstract dynamics $\hat{\bar{p}}\left(\bar{s}' \mid \bar{s}, \bar{a}\right)$ are implemented as a neural network that predicts the weights, means, and standard deviations of a mixture of Gaussians. In addition, we estimate the probability of episode termination via another head. Both are trained via maximum likelihood. The second model, $\hat{\bar{r}}_\pi\left(\bar{s}, \bar{a}\right)$, predicts the reward that accumulates during the execution of a skill and is trained with a mean squared error loss. The termination probability is used during planning to correctly weight rewards that are less likely to occur because the episode might terminate beforehand.

C Baselines

We implemented **DADS** with a custom version of TD-MPC2 to enable a fair comparison to SPlaTES. As with our method, we added (i) the skill vector which is unchanged during rollouts, and (ii) support for vectorized environments. As the MPPI planner of TD-MPC2 does not provide probabilities for actions, we could not implement the offline version of DADS (Sharma et al., 2020a). However, we think that the gains in sample efficiency from using model-based TD-MPC2 outweigh the disadvantage of not being able to use importance sampling.

We combined **HER** with TD-MPC2 in a similar way, by keeping track of the goal instead of the skill vector. Hindsight relabeling was implemented to take place during sampling from the replay buffer.

D Experimental details

D.1 Environments

Figure 14 shows the environments used for the experiments in Section 5. Ant Maze Push and Fetch P & P Barrier are new variations and can be found on the project page.

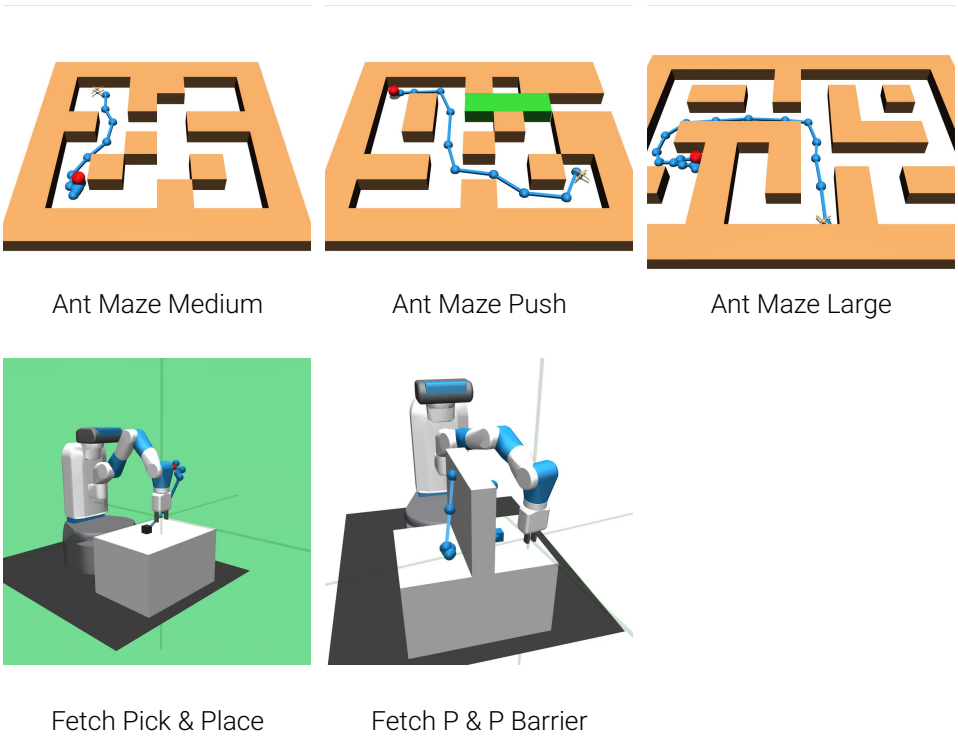


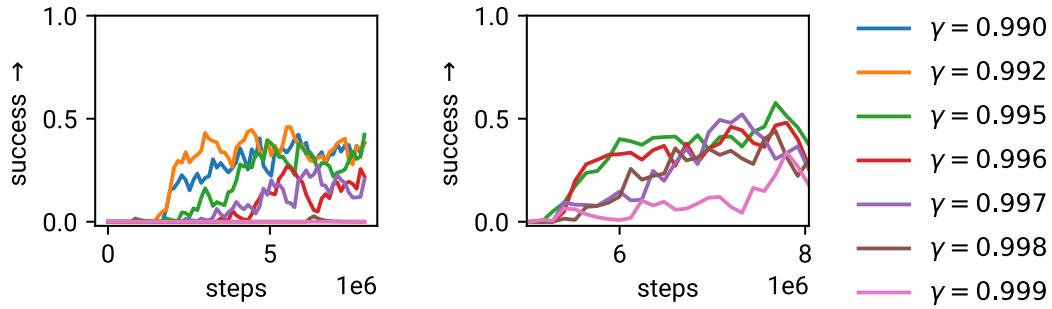
Figure 14: Renderings of the MuJoCo environments used in Section 5

D.2 Number of seeds and reported quantities

The results in figures 6, 7, 8, and 11 were obtained by repeating training with five different seeds for SPlaTES and all baselines. Lines (or bars) indicate the median, whereas the shaded area (or error bars) indicate the range between the 20% and 80% percentiles. Five seeds were chosen as the number is high enough to clearly separate our method from the baselines and ablations (except on Fetch Pick & Place which is solved by all algorithms), while still fitting into our compute budget.

D.3 Hyperparameter studies

This section contains information about how hyperparameters of SPlaTES and the baselines influence performance.



(a) TD-MPC2 (from scratch) (b) TD-MPC2 (distilled from SPlaTES)

Figure 15: (a) Grid search over discount factor γ for TD-MPC2 when training from scratch. (b) Grid search over discount factor when distilling a SPlaTES agent into TD-MPC2 model as described in Section 5.3.

Figure 15 (a) shows the success rates of TD-MPC2 for a range of discount factors on Ant Maze Medium. In principle, larger discount factors can enable less myopic behavior as they put more emphasis on rewards far in the future. However, we observed empirically that discount factors $\gamma > 0.995$ have a negative impact on performance. This is probably caused by the instability of TD-learning with discount factors close to 1, in particular in combination with function approximation, and off-policy training (Sutton et al., 1998). We chose $\gamma = 0.995$ for our experiments as it is the highest discount factor for which we still observed good asymptotic performance.

Figure 15 (b) shows the performance of TD-MPC2 agents being distilled from a SPlaTES agent (see Section 5.2) for various discount factors. Again we observe that discount factors $\gamma > 0.995$ perform worse. However, the performance of high discount factors is still better than when training TD-MPC2 from scratch. This can probably be attributed to two factors: (i) having access to the replay buffer of the SPlaTES agent, and (ii) being transported to different locations in the environment by this agent. This provides a better data distribution and stabilizes training. We again chose $\gamma = 0.995$ for the experiments in the main text as this discount factor performed best.

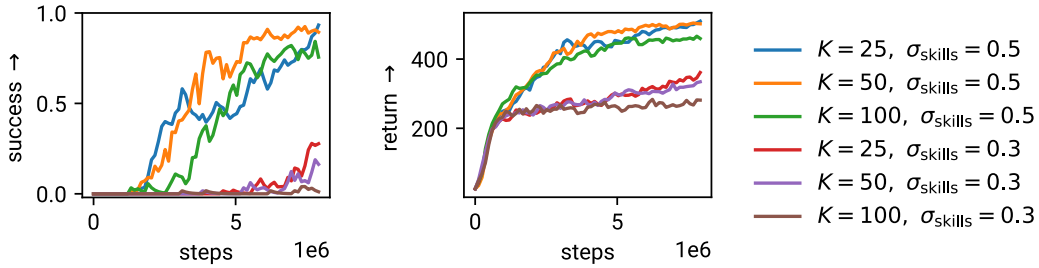


Figure 16: Grid search over skill length K of SPlaTES and fixed standard deviation σ_{skills} of the abstract dynamics model used for computing the skill reward.

Figure 16 shows a grid search over the skill length K of SPlaTES and the fixed standard deviation σ_{skills} of the abstract dynamics model used for computing the skill reward. Choosing σ_{skills} too small removes the incentive for the skills to fill as much of the abstract state space as possible, and leads to slow learning. The length K of a skill execution, on the other hand, does not have a big influence on the final performance (as long as it is big enough to enable significant temporal abstraction). We therefore chose $K = 5$ and $\sigma_{\text{skills}} = 0.5$.

D.4 Hyperparameters

Table 1 contains important hyperparameters for SPlaTES. These are:

- the **skill duration** K, which determines for how many time steps a skill is executed,

- the **dimension $d^{\bar{\mathcal{A}}}$ of the skill vector/abstract action** $\bar{a}$,
- the **dimension $d^{\bar{\mathcal{S}}}$ of the abstract state** $\bar{s}$,
- the **fixed standard deviation σ_{skills}** of the abstract dynamics model which controls the scale over which the skills repel each other in the abstract state space,
- the **symmetry breaking duration**, during which a random linear dynamics model is used in the intrinsic reward instead of the trained one (after half of the duration, a linear interpolation of the two models is used that linearly reduces the weight of the symmetry breaking model until the end of the symmetry breaking phase), and
- the **reward clipping duration**, during which the intrinsic reward is clipped at zero from below.

Hyperparameter	**Environment**	
	Fetch variations	Ant variations
K	10	50
skill dim. $d^{\bar{\mathcal{A}}}$	3	2
abstract state dim. $d^{\bar{\mathcal{S}}}$	3	2
σ_{skills}	$1.0 \rightarrow 0.3$	(Push variant: $1.5 \rightarrow$) 0.5
symmetry breaking duration	300k	200k
reward clipping duration	160k	-

Table 1: Important SPlaTES hyperparameters. $a \rightarrow b$ indicates a schedule over the training. This table focuses on hyperparameters that have a big impact on the skill learning or the hierarchy as a whole. We refer to the config files in the code repository for a complete list of hyperparameters (concerning TD-MPC2, MLP sizes etc.)