Temporal Difference Learning for Model Predictive Control

Nicklas Hansen [1] **Xiaolong Wang** [*1] **Hao Su** [*1]

Abstract

Data-driven model predictive control has two key advantages over model-free methods: a potential for improved sample efficiency through model learning, and better performance as computational budget for planning increases. However, it is both costly to plan over long horizons and challenging to obtain an accurate model of the environment. In this work, we combine the strengths of model-free and model-based methods. We use a learned task-oriented latent dynamics model for local trajectory optimization over a short horizon, and use a learned terminal value function to estimate long-term return, both of which are *learned jointly* by temporal difference learning. Our method, **TD-MPC**, achieves superior sample efficiency and asymptotic performance over prior work on both state and image-based continuous control tasks from DMControl and Meta-World. Code and videos are available at https://nicklashansen.github.io/td-mpc.

1. Introduction

To achieve desired behavior in an environment, a Reinforcement Learning (RL) agent needs to iteratively interact and consolidate knowledge about the environment. Planning is a powerful approach to such sequential decision making problems, and has achieved tremendous success in application areas such as game-playing (Kaiser et al., 2020; Schrittwieser et al., 2020) and continuous control (Tassa et al., 2012; Chua et al., 2018; Janner et al., 2019). By utilizing an internal model of the environment, an agent can plan a trajectory of actions ahead of time that leads to the desired behavior; this is in contrast to *model-free* algorithms that learn a policy purely through trial-and-error.

Concretely, prior work on model-based methods can largely be subdivided into two directions, each exploiting key advantages of model-based learning: *(i)* planning, which is advantageous over a learned policy, but it can be prohibitively expensive to plan over long horizons (Janner et al., 2019; Lowrey et al., 2019; Hafner et al., 2019; Argenson & Dulac-Arnold, 2021); and *(ii)* using a learned model to improve sample-efficiency of model-free methods by e.g. learning from generated rollouts, but this makes model biases likely to propagate to the policy as well (Ha & Schmidhuber, 2018; Hafner et al., 2020b; Clavera et al., 2020). As a result, model-based methods have historically struggled to outperform simpler, model-free methods (Srinivas et al., 2020; Kostrikov et al., 2020) in continuous control tasks.

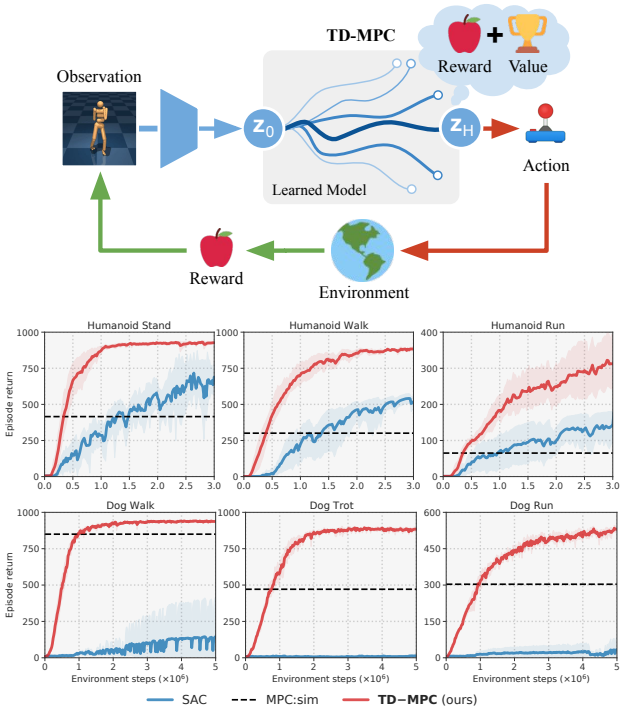

Figure 1. **Overview.** *(Top)* We present a framework for MPC using a task-oriented latent dynamics model and value function learned jointly by temporal difference learning. We perform trajectory optimization over model rollouts and use the value function for long-term return estimates. *(Bottom)* Episode return of our method, SAC, and MPC with a ground-truth simulator on challenging, high-dimensional Humanoid and Dog tasks (Tassa et al., 2018). Mean of 5 runs; shaded areas are 95% confidence intervals.

Can we instead augment model-based planning with the strengths of model-free learning? Because of the immense cost of long-horizon planning, Model Predictive Control

[*]Equal contribution [1]UC San Diego. Correspondence to: Nicklas Hansen <nihansen@ucsd.edu>.

Proceedings of the 39th International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022.

(MPC) optimizes a trajectory over a shorter, finite horizon, which yields only temporally local optimal solutions. MPC can be extended to approximate globally optimal solutions by using a terminal value function that estimates discounted return beyond the planning horizon. However, obtaining an accurate model and value function can be challenging.

In this work, we propose **T**emporal **D**ifference Learning for **M**odel **P**redictive **C**ontrol (**TD-MPC**), a framework for data-driven MPC using a task-oriented latent dynamics model and terminal value function *learned jointly* by temporal difference (TD) learning. At each decision step, we perform trajectory optimization using short-term reward estimates generated by the learned model, and use the learned value function for long-term return estimates. For example, in the Humanoid locomotion task shown in Figure 1, planning with a model may be beneficial for accurate joint movement, whereas the higher-level objective, e.g. direction of running, can be guided by long-term value estimates.

A key technical contribution is how the model is learned. While prior work learns a model through state or video prediction, we argue that it is remarkably inefficient to model everything in the environment, including irrelevant quantities and visuals such as shading, as this approach suffers from model inaccuracies and compounding errors. To overcome these challenges, we make three key changes to model learning. Firstly, we learn the latent representation of the dynamics model purely from rewards, ignoring nuances unnecessary for the task at hand. This makes the learning more sample efficient than state/image prediction. Secondly, we back-propagate gradients from the reward and TD-objective through multiple rollout steps of the model, improving reward and value predictions over long horizons. This alleviates error compounding when conducting rollouts. Lastly, we propose a modality-agnostic prediction loss *in latent space* that enforces temporal consistency in the learned representation *without* explicit state or image prediction.

We evaluate our method on a variety of continuous control tasks from DMControl (Tassa et al., 2018) and Meta-World (Yu et al., 2019), where we find that our method achieves superior sample efficiency and asymptotic performance over prior model-based and model-free methods. In particular, our method solves Humanoid and Dog locomotion tasks with up to 38-dimensional continuous action spaces in as little as 1M environment steps (see Figure 1), and is trivially extended to match the state-of-the-art in image-based RL.

2. Preliminaries

Problem formulation. We consider infinite-horizon Markov Decision Processes (MDP) characterized by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma, p_0)$, where $\mathcal{S} \in \mathbb{R}^n$ and $\mathcal{A} \in \mathbb{R}^m$ are continuous state and action spaces, $\mathcal{T}\colon \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}_+$ is the transition (dynamics) function, $\mathcal{R}\colon \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ is a reward function, $\gamma \in [0, 1)$ is a discount factor, and p_0 is the initial state distribution. We aim to learn a parameterized mapping $\Pi_\theta\colon \mathcal{S} \mapsto \mathcal{A}$ with parameters θ such that discounted return $\mathbb{E}_{\Gamma\sim\Pi_\theta}[\sum_{t=1}^{\infty} \gamma^t r_t],\ r_t \sim \mathcal{R}(\cdot|\mathbf{s}_t, \mathbf{a}_t)$ is maximized along a trajectory $\Gamma = (\mathbf{s}_0, \mathbf{a}_0, \mathbf{s}_1, \mathbf{a}_1, \dots)$ following Π_θ by sampling an action $\mathbf{a}_t \sim \Pi_\theta(\cdot|\mathbf{s}_t)$ and reaching state $\mathbf{s}_{t+1} \sim \mathcal{T}(\cdot|\mathbf{s}_t, \mathbf{a}_t)$ at each decision step t.

Fitted Q-iteration. Model-free TD-learning algorithms aim to estimate an optimal state-action value function $Q^*\colon \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ using a parametric value function $Q_\theta(\mathbf{s}, \mathbf{a}) \approx Q^*(\mathbf{s}, \mathbf{a}) = \max_{\mathbf{a}'} \mathbb{E}[\mathcal{R}(\mathbf{s}, \mathbf{a}) + \gamma Q^*(\mathbf{s}', \mathbf{a}')]\ \forall \mathbf{s} \in \mathcal{S}$ where $\mathbf{s}', \mathbf{a}'$ is the state and action at the following step, and θ parameterizes the function (Sutton, 2005). For $\gamma \approx 1$, Q^* estimates discounted return for the optimal policy over an infinite horizon. While Q^* is generally unknown, it can be approximated by repeatedly fitting Q_θ using the update rule

$$\theta^{k+1} \leftarrow \arg\min_\theta \mathbb{E}_{(\mathbf{s},\mathbf{a},\mathbf{s}')\sim\mathcal{B}} \left\| Q_\theta(\mathbf{s}, \mathbf{a}) - y \right\|_2^2 \qquad (1)$$

where the Q-*target* $y = \mathcal{R}(\mathbf{s}, \mathbf{a}) + \gamma \max_{\mathbf{a}'} Q_{\theta^-}(\mathbf{s}', \mathbf{a}')$, $\mathcal{B}$ is a replay buffer that is iteratively grown as new data is collected, and θ^- is a slow-moving average of the online parameters θ updated with the rule $\theta^-_{k+1} \longleftarrow (1-\zeta)\theta^-_k + \zeta\theta_k$ at each iteration using a constant coefficient $\zeta \in [0, 1)$.

Model Predictive Control. In actor-critic RL algorithms, Π is typically a policy parameterized by a neural network that learns to approximate $\Pi_\theta(\cdot|\mathbf{s}) \approx \arg\max_{\mathbf{a}} \mathbb{E}[Q_\theta(\mathbf{s}, \mathbf{a})]\ \forall \mathbf{s} \in \mathcal{S}$, i.e, the globally optimal policy. In control, Π is traditionally implemented as a trajectory optimization procedure. To make the problem tractable, one typically obtains a *local* solution to the trajectory optimization problem at each step t by estimating optimal actions $\mathbf{a}_{t:t+H}$ over a finite horizon H and executing the first action $\mathbf{a}_t$, known as *Model Predictive Control* (MPC):

$$\Pi^{\text{MPC}}_\theta(\mathbf{s}_t) = \arg\max_{\mathbf{a}_{t:t+H}} \mathbb{E}\left[\sum_{i=t}^{H} \gamma^i \mathcal{R}(\mathbf{s}_i, \mathbf{a}_i)\right], \qquad (2)$$

where γ, unlike in fitted Q-iteration, is typically set to 1, i.e., no discounting. Intuitively, Equation 2 can be viewed as a special case of the standard additive-cost optimal control objective. A solution can be found by iteratively fitting parameters of a family of distributions, e.g., μ, σ for a multivariate Gaussian with diagonal covariance, to the space of actions over a finite horizon using the derivative-free Cross-Entropy Method (CEM; Rubinstein (1997)), and sample trajectories generated by a model. As opposed to fitted Q-iteration, Equation 2 is not predictive of long-term rewards, hence a *myopic* solution. When a value function is known (e.g. a heuristic or in the context of our method: estimated using Equation 1), it can be used in conjunction with Equation 2 to estimate discounted return at state $\mathbf{s}_{t+H}$ and

beyond; such methods are known as MPC with a *terminal value function*. In the following, we consider parameterized mappings Π from both the perspective of actor-critic RL algorithms and model predictive control (planning). To disambiguate these concepts, we refer to planning with MPC as Π_θ and a policy network as π_θ. We generically denote parameterization using neural networks as θ (online) and θ^- (*target*; slow-moving average of θ) as combined feature vectors.

3. TD-Learning for Model Predictive Control

We propose **TD-MPC**, a framework that combines MPC with a task-oriented latent dynamics model and terminal value function jointly learned using TD-learning in an online RL setting. Specifically, TD-MPC leverages Model Predictive Path Integral (MPPI; Williams et al. (2015)) control for planning (denoted Π_θ), learned models d_θ, R_θ of the (latent) dynamics and reward signal, respectively, a terminal state-action value function Q_θ, and a parameterized policy π_θ that helps guide planning. We summarize our framework in Figure 1 and Algorithm 1. In this section, we detail the inference-time behavior of our method, while we defer discussion of training to Section 4.

MPPI is an MPC algorithm that iteratively updates parameters for a family of distributions using an importance weighted average of the estimated top-k sampled trajectories (in terms of expected return); in practice, we fit parameters of a time-dependent multivariate Gaussian with diagonal covariance. We adapt MPPI as follows. Starting from initial parameters $(\mu^0, \sigma^0)_{t:t+H},\ \mu^0, \sigma^0 \in \mathbb{R}^m,\ \mathcal{A} \in \mathbb{R}^m$, i.e. independent parameters for each action over a horizon of length H, we independently sample N trajectories using rollouts generated by the learned model d_θ, and estimate the total return ϕ_Γ of a sampled trajectory Γ as

$$\phi_\Gamma \triangleq \mathbb{E}_\Gamma \left[\gamma^H Q_\theta(\mathbf{z}_H, \mathbf{a}_H) + \sum_{t=0}^{H-1} \gamma^t R_\theta(\mathbf{z}_t, \mathbf{a}_t) \right], \quad (3)$$

where $\mathbf{z_{t+1}} = d_\theta(\mathbf{z}_t, \mathbf{a}_t)$ and $\mathbf{a}_t \sim \mathcal{N}(\mu_t^{j-1}, (\sigma_t^{j-1})^2 \mathrm{I})$ at iteration $j-1$, as highlighted in red in Algorithm 1. We select the top-k returns $\phi_\Gamma^\star$ and obtain new parameters μ^j, σ^j at iteration j from a $\phi_\Gamma^\star$-normalized empirical estimate:

$$\mu^j = \frac{\sum_{i=1}^k \Omega_i \Gamma_i^\star}{\sum_{i=1}^k \Omega_i},\ \sigma^j = \sqrt{\frac{\sum_{i=1}^k \Omega_i (\Gamma_i^\star - \mu^j)^2}{\sum_{i=1}^k \Omega_i}}, \quad (4)$$

where $\Omega_i = e^{\tau(\phi_{\Gamma,i}^\star)}$, τ is a temperature parameter controlling the "sharpness" of the weighting, and $\Gamma_i^\star$ denotes the ith top-k trajectory corresponding to return estimate $\phi_\Gamma^\star$. After a fixed number of iterations J, the planning procedure is terminated and a trajectory is sampled from the final return-normalized distribution over action sequences. We plan at each decision step t and execute only the first action, i.e., we employ *receding-horizon* MPC to produce a feedback policy. To reduce the number of iterations required for convergence, we "warm start" trajectory optimization at each step t by reusing the 1-step shifted mean μ obtained at the previous step (Argenson & Dulac-Arnold, 2021), but always use a large initial variance to avoid local minima.

Algorithm 1 TD-MPC *(inference)*

Require: θ : learned network parameters
μ^0, σ^0: initial parameters for $\mathcal{N}$
N, N_π: num sample/policy trajectories
$\mathbf{s}_t, H$: current state, rollout horizon

Encode state $\mathbf{z}_t \leftarrow h_\theta(\mathbf{s}_t)$ ◁ *Assuming TOLD model*
for each iteration $j = 1..J$ **do**
  Sample N traj. of len. H from $\mathcal{N}(\mu^{j-1}, (\sigma^{j-1})^2 \mathrm{I})$
  Sample N_π traj. of length H using π_θ, d_θ
  // Estimate trajectory returns ϕ_Γ using $d_\theta, R_\theta, Q_\theta$, starting from $\mathbf{z}_t$ and initially letting $\phi_\Gamma = 0$:
  for all $N + N_\pi$ trajectories $(\mathbf{a}_t, \mathbf{a}_{t+1}, \ldots, \mathbf{a}_{t+H})$ **do**
    for step $t = 0..H-1$ **do**
      $\phi_\Gamma = \phi_\Gamma + \gamma^t R_\theta(\mathbf{z}_t, \mathbf{a}_t)$ ◁ *Reward*
      $\mathbf{z}_{t+1} \leftarrow d_\theta(\mathbf{z}_t, \mathbf{a}_t)$ ◁ *Latent transition*
    $\phi_\Gamma = \phi_\Gamma + \gamma^H Q_\theta(\mathbf{z}_H, \mathbf{a}_H)$ ◁ *Terminal value*
  // Update parameters μ, σ for next iteration:
  $\mu^j, \sigma^j =$ Equation 4 (and Equation 5)
return $\mathbf{a} \sim \mathcal{N}(\mu^J, (\sigma^J)^2 \mathrm{I})$

Exploration by planning. Model-free RL algorithms such as DDPG (Lillicrap et al., 2016) encourage exploration by injecting action noise (e.g. Gaussian or Ornstein-Uhlenbeck noise) into the learned policy π_θ during training, optionally following a linear annealing schedule. While our trajectory optimization procedure is inherently stochastic due to trajectory sampling, we find that the rate at which σ decays varies wildly between tasks, leading to (potentially poor) local optima for small σ. To promote consistent exploration across tasks, we constrain the std. deviation of the sampling distribution such that, for a μ^j obtained from Equation 4 at iteration j, we instead update σ^j to

$$\sigma^j = \max\left(\sqrt{\frac{\sum_{i=1}^N \Omega_i (\Gamma_i^\star - \mu^j)^2}{\sum_{i=1}^N \Omega_i}},\ \epsilon \right), \quad (5)$$

where $\epsilon \in \mathbb{R}_+$ is a linearly decayed constant. Likewise, we linearly increase the planning horizon from 1 to H in the early stages of training, as the model is initially inaccurate and planning would therefore be dominated by model bias.

Policy-guided trajectory optimization. Analogous to Schrittwieser et al. (2020); Sikchi et al. (2022), TD-MPC learns a policy π_θ in addition to planning procedure Π_θ, and augments the sampling procedure with additional samples from π_θ (highlighted in blue in Algorithm 1). This leads to

one of two cases: the policy trajectory is estimated to be *(i)* poor, and may be *excluded* from the top-k trajectories; or *(ii)* good, and may be *included* with influence proportional to its estimated return ϕ_Γ. While LOOP relies on the maximum entropy objective of SAC (Haarnoja et al., 2018) for exploration, TD-MPC learns a deterministic policy. To make sampling stochastic, we apply linearly annealed (Gaussian) noise to π_θ actions as in DDPG (Lillicrap et al., 2016). Our full procedure is summarized in Algorithm 1.

4. Task-Oriented Latent Dynamics Model

To be used in conjunction with TD-MPC, we propose a **T**ask-**O**riented **L**atent **D**ynamics (TOLD) model that is jointly learned together with a terminal value function using TD-learning. Rather than attempting to model the environment itself, our TOLD model learns to only model elements of the environment that are predictive of reward, which is a far easier problem. During inference, our TD-MPC framework leverages the learned TOLD model for trajectory optimization, estimating short-term rewards using model rollouts and long-term returns using the terminal value function. TD-MPC and TOLD support continuous action spaces, arbitrary input modalities, and sparse reward signals. Figure 2 provides an overview of the TOLD training procedure.

Components. Throughout training, our agent iteratively performs the following two operations: *(i)* improving the learned TOLD model using data collected from previous environment interaction; and *(ii)* collecting new data from the environment by online planning of action sequences with TD-MPC, using TOLD for generating imagined rollouts. Our proposed TOLD consists of five learned components $h_\theta, d_\theta, R_\theta, Q_\theta, \pi_\theta$ that predict the following quantities:

$$\begin{aligned} &\text{Representation:} && \mathbf{z}_t = h_\theta(\mathbf{s}_t) \\ &\text{Latent dynamics:} && \mathbf{z}_{t+1} = d_\theta(\mathbf{z}_t, \mathbf{a}_t) \\ &\text{Reward:} && \hat{r}_t = R_\theta(\mathbf{z}_t, \mathbf{a}_t) \\ &\text{Value:} && \hat{q}_t = Q_\theta(\mathbf{z}_t, \mathbf{a}_t) \\ &\text{Policy:} && \hat{\mathbf{a}}_t \sim \pi_\theta(\mathbf{z}_t) \end{aligned} \tag{6}$$

Given an observation $\mathbf{s}_t$ observed at time t, a representation network h_θ encodes $\mathbf{s}_t$ into a latent representation $\mathbf{z}_t$. From $\mathbf{z}_t$ and an action $\mathbf{a}_t$ taken at time t, TOLD then predicts *(i)* the latent dynamics (latent representation $\mathbf{z}_{t+1}$ of the following timestep); *(ii)* the single-step reward received; *(iii)* its state-action (Q) value; and *(iv)* an action that (approximately) maximizes the Q-function. To make TOLD less susceptible to compounding errors, we recurrently predict the aforementioned quantities multiple steps into the future from predicted future latent states, and back-propagate gradients through time. Unlike prior work (Ha & Schmidhuber, 2018; Janner et al., 2019; Hafner et al., 2019; 2020b; Sikchi et al., 2022), we find it sufficient to implement all components of TOLD as purely deterministic MLPs, i.e., *without* RNN gating mechanisms nor probabilistic models.

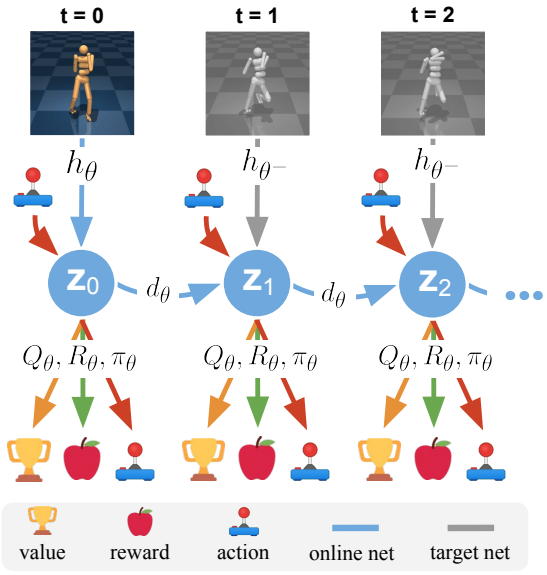

Figure 2. **Training our TOLD model.** A trajectory $\Gamma_{0:H}$ of length H is sampled from a replay buffer, and the first observation $\mathbf{s}_0$ is encoded by h_θ into a latent representation $\mathbf{z}_0$. Then, TOLD recurrently predicts the following latent states $\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_H$, as well as a value $\hat{q}$, reward $\hat{r}$, and action $\hat{\mathbf{a}}$ for each latent state, and we optimize TOLD using Equation 7. Subsequent observations are encoded using target net h_{θ^-} (θ^-: slow-moving average of θ) and used as latent targets only during training (illustrated in gray).

Objective. We first state the full objective, and then motivate each module and associated objective term. During training, we minimize a temporally weighted objective

$$\mathcal{J}(\theta; \Gamma) = \sum_{i=t}^{t+H} \lambda^{i-t} \mathcal{L}(\theta; \Gamma_i)\,, \tag{7}$$

where $\Gamma \sim \mathcal{B}$ is a trajectory $(\mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}_{t+1})_{t:t+H}$ sampled from a replay buffer $\mathcal{B}$, $\lambda \in \mathbb{R}_+$ is a constant that weights near-term predictions higher, and the single-step loss

$$\mathcal{L}(\theta; \Gamma_i) = c_1 \underbrace{\|R_\theta(\mathbf{z}_i, \mathbf{a}_i) - r_i\|_2^2}_{\text{reward}} \tag{8}$$

$$+ c_2 \underbrace{\|Q_\theta(\mathbf{z}_i, \mathbf{a}_i) - (r_i + \gamma Q_{\theta^-}(\mathbf{z}_{i+1}, \pi_\theta(\mathbf{z}_{i+1})))\|_2^2}_{\text{value}} \tag{9}$$

$$+ c_3 \underbrace{\|d_\theta(\mathbf{z}_i, \mathbf{a}_i) - h_{\theta^-}(\mathbf{s}_{i+1})\|_2^2}_{\text{latent state consistency}} \tag{10}$$

is employed to jointly optimize for reward prediction, value prediction, and a latent state consistency loss that regularizes the learned representation. Here, $c_{1:3}$ are constant coefficients balancing the three losses. From each transition $(\mathbf{z}_i, \mathbf{a}_i)$, the reward term (Equation 8) predicts the single-step reward, the value term (Equation 9) is our adoption of fitted Q-iteration from Equation 1 following previous work on actor-critic algorithms (Lillicrap et al., 2016; Haarnoja et al., 2018), and the consistency term (Equation 10) predicts the latent representation of future states.

Crucially, recurrent predictions are made entirely in latent space from states $\mathbf{z}_i = h_\theta(\mathbf{s}_i)$, $\mathbf{z}_{i+1} = d_\theta(\mathbf{z}_i, \mathbf{a}_i)$, $\dots$, $\mathbf{z}_{i+H} = d_\theta(\mathbf{z}_{i+H-1}, \mathbf{a}_{i+H-1})$ such that only the first observation $\mathbf{s}_i$ is encoded using h_θ and gradients *from all three terms* are back-propagated *through time*. This is in contrast to prior work on model-based learning that learn a model by state or video prediction, entirely decoupled from policy and/or value learning (Ha & Schmidhuber, 2018; Hafner et al., 2020b; Sikchi et al., 2022). We use an exponential moving average θ^- of the online network parameters θ for computing the value target (Lillicrap et al., 2016), and similarly also use θ^- for the latent state consistency target $h_{\theta^-}(\mathbf{s}_{i+1})$. The policy π_θ is described next, while we defer discussion of the consistency loss to the following section.

Computing TD-targets. The TD-objective in Equation 9 requires estimating the quantity $\max_{\mathbf{a}_t} Q_{\theta^-}(\mathbf{z}_t, \mathbf{a}_t)$, which is extremely costly to compute using planning (Lowrey et al., 2019). Therefore, we instead learn a policy π_θ that maximizes Q_θ by minimizing the objective

$$\mathcal{J}_\pi(\theta; \Gamma) = -\sum_{i=t}^{t+H} \lambda^{i-t} Q_\theta(\mathbf{z}_i, \pi_\theta(\text{sg}(\mathbf{z}_i))) \,, \tag{11}$$

which is a temporally weighted adaptation of the policy objective commonly used in model-free actor-critic methods such as DDPG (Lillicrap et al., 2016) and SAC (Haarnoja et al., 2018). Here, sg denotes the stop-grad operator, and Equation 11 is optimized only wrt. policy parameters. While we empirically observe that for complex tasks the learned π_θ is inferior to planning (discussed in Section 5), we find it sufficiently expressive for efficient value learning.

Latent state consistency. To provide a rich learning signal for model learning, prior work on model-based RL commonly learn to directly predict future states or pixels (Ha & Schmidhuber, 2018; Janner et al., 2019; Lowrey et al., 2019; Kaiser et al., 2020; Sikchi et al., 2022). However, learning to predict future observations is an extremely hard problem as it forces the network to model everything in the environment, including task-irrelevant quantities and details such as shading. Instead, we propose to regularize TOLD with a *latent state consistency loss* (shown in Equation 10) that forces a future latent state prediction $\mathbf{z}_{t+1} = d_\theta(\mathbf{z}_t, \mathbf{a}_t)$ at time $t+1$ to be similar to the latent representation of the corresponding ground-truth observation $h_{\theta^-}(\mathbf{s}_{t+1})$, circumventing prediction of observations altogether. Additionally, this design choice effectively makes model learning agnostic to the observation modality. The training procedure is shown in Algorithm 2; see Appendix F for pseudo-code.

Algorithm 2 TOLD *(training)*

Require: θ, θ^-: randomly initialized network parameters
$\eta, \tau, \lambda, \mathcal{B}$: learning rate, coefficients, buffer

while not tired **do**
  // Collect episode with TD-MPC from $\mathbf{s}_0 \sim p_0$:
  for step $t = 0 ... T$ **do**
    $\mathbf{a}_t \sim \Pi_\theta(\cdot | h_\theta(\mathbf{s}_t))$ ◁ *Sample with TD-MPC*
    $(\mathbf{s}_{t+1},\ r_t) \sim \mathcal{T}(\cdot|\mathbf{s}_t, \mathbf{a}_t),\ \mathcal{R}(\cdot|\mathbf{s}_t, \mathbf{a}_t)$ ◁ *Step env.*
    $\mathcal{B} \leftarrow \mathcal{B} \cup (\mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}_{t+1})$ ◁ *Add to buffer*
  // Update TOLD using collected data in $\mathcal{B}$:
  for num updates per episode **do**
    $\{\mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}_{t+1}\}_{t:t+H} \sim \mathcal{B}$ ◁ *Sample traj.*
    $\mathbf{z}_t = h_\theta(\mathbf{s}_t)$ ◁ *Encode first observation*
    $J = 0$ ◁ *Initialize J for loss accumulation*
    for $i = t ... t+H$ **do**
      $\hat{r}_i = R_\theta(\mathbf{z}_i, \mathbf{a}_i)$ ◁ *Equation 8*
      $\hat{q}_i = Q_\theta(\mathbf{z}_i, \mathbf{a}_i)$ ◁ *Equation 9*
      $\mathbf{z}_{i+1} = d_\theta(\mathbf{z}_i, \mathbf{a}_i)$ ◁ *Equation 10*
      $\hat{\mathbf{a}}_i = \pi_\theta(\mathbf{z}_i)$ ◁ *Equation 11*
      $J \leftarrow J + \lambda^{i-t} \mathcal{L}(\mathbf{z}_{i+1}, \hat{r}_i, \hat{q}_i, \hat{\mathbf{a}}_i)$ ◁ *Equation 7*
    $\theta \leftarrow \theta - \frac{1}{H}\eta \nabla_\theta J$ ◁ *Update online network*
    $\theta^- \leftarrow (1-\tau)\theta^- + \tau\theta$ ◁ *Update target network*

5. Experiments

We evaluate TD-MPC with a TOLD model on a total of **92** diverse and challenging continuous control tasks from DeepMind Control Suite (DMControl; Tassa et al. (2018)) and Meta-World *v2* (Yu et al., 2019), including tasks with sparse rewards, high-dimensional state and action spaces, image observations, multi-modal inputs, goal-conditioning, and multi-task learning settings; see Appendix L for task visualizations. We choose these two benchmarks for their great task diversity and availability of baseline implementations and results. We seek to answer the following questions:

— How does planning with TD-MPC compare to state-of-the-art model-based and model-free approaches?

— Are TOLD models capable of multi-task and transfer behaviors despite using a reward-centric objective?

— How does performance relate to the computational budget of the planning procedure?

An implementation of TD-MPC is available at `https://nicklashansen.github.io/td-mpc`, which will solve most tasks in an hour on a single GPU.

Implementation details. All components are deterministic and implemented using MLPs. We linearly anneal the exploration parameter ϵ of Π_θ and π_θ from 0.5 to 0.05 over the first 25k decision steps[1]. We use a planning horizon of $H = 5$, and sample trajectories using prioritized experience replay (Schaul et al., 2016) with priority scaled by the value loss. During planning, we plan for 6 iterations (8 for Dog; 12 for Humanoid), sampling $N = 512$ trajectories ($+5\%$ sampled from π_θ), and we compute μ, σ parameters over

[1]To avoid ambiguity, we refer to simulation steps as *environment steps* (independent of action repeat), and use *decision steps* when referring to policy queries (dependent on action repeat).

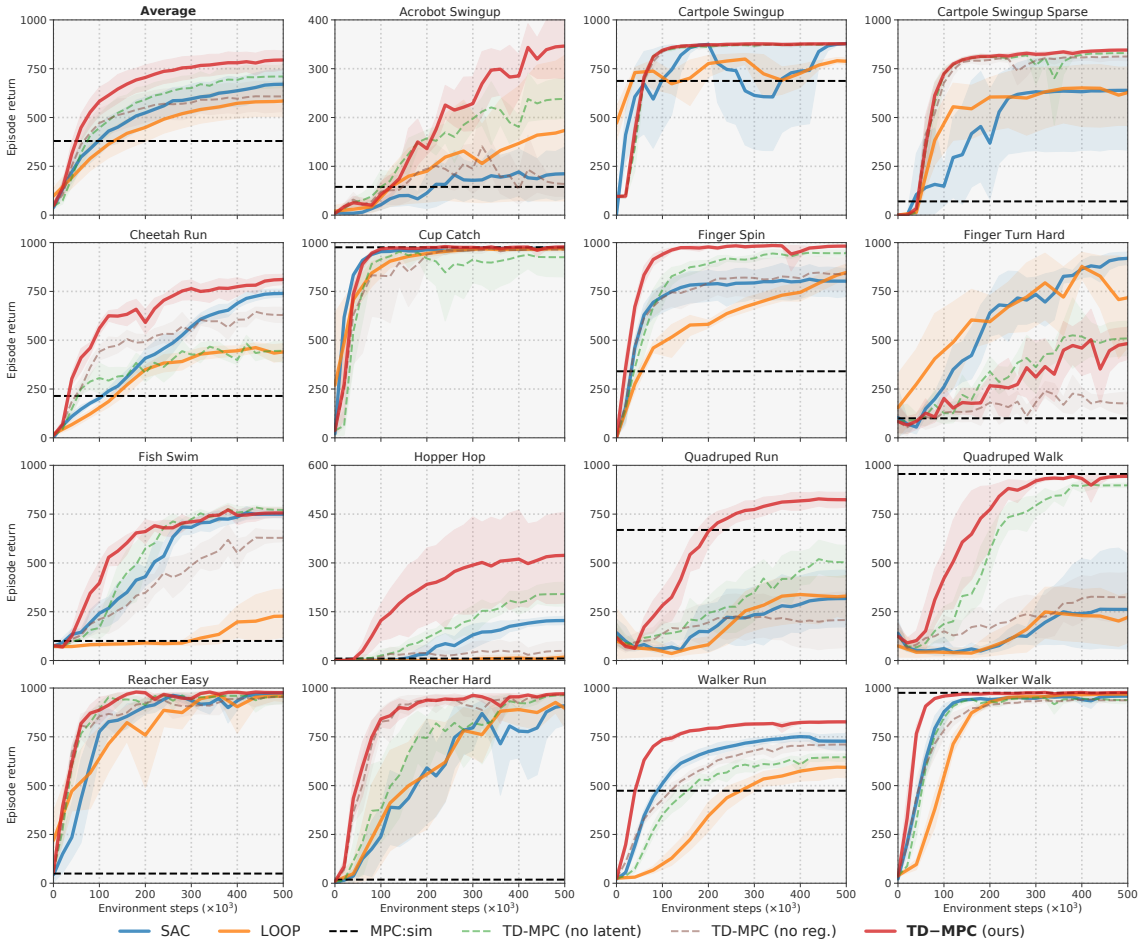

Figure 3. **DMControl tasks.** Return of our method (TD-MPC) and baselines on 15 state-based continuous control tasks from DMControl (Tassa et al., 2018). Mean of 5 runs; shaded areas are 95% confidence intervals. In the top left, we visualize results averaged across all 15 tasks. We observe especially large performance gains on tasks with complex dynamics, e.g., the *Quadruped* and *Acrobot* tasks.

the top-64 trajectories each iteration. For image-based tasks, observations are 3 stacked 84×84-dimensional RGB frames and we use ± 4 pixel shift augmentation (Kostrikov et al., 2020). Refer to Appendix F for additional details.

Baselines. We evaluate our method against the following:

— **Soft Actor-Critic** (SAC; (Haarnoja et al., 2018)), a state-of-the-art model-free algorithm derived from maximum entropy RL (Ziebart et al., 2008). We choose SAC as our main point of comparison due to its popularity and strong performance on both DMControl and Meta-World. In particular, we adopt the implementation of Yarats & Kostrikov (2020).

— **LOOP** (Sikchi et al., 2022), a hybrid algorithm that extends SAC with planning and a learned model. LOOP has been shown to outperform a number of model-based methods, e.g., MBPO (Janner et al., 2019) and POLO (Lowrey et al., 2019)) on select MuJoCo tasks. It is a particularly relevant baseline due to its similarities to TD-MPC.

— **MPC** with a *ground-truth* simulator (denoted *MPC:sim*). As planning with a simulator is computationally intensive, we limit the planning horizon to 10 ($2\times$ ours), sampled trajectories to 200, and optimize for 4 iterations (ours: 6).

— **CURL** (Srinivas et al., 2020), **DrQ** (Kostrikov et al., 2020), and **DrQ-v2** (Yarats et al., 2021), three state-of-the-art model-free algorithms.

— **PlaNet** (Hafner et al., 2019), **Dreamer** (Hafner et al., 2020b), and **Dreamer-v2** (Hafner et al., 2020a). All three methods learn a model using a reconstruction loss, and select actions using either MPC or a learned policy.

— **MuZero** (Schrittwieser et al., 2020) and **EfficientZero** (Ye et al., 2021), which learn a latent dynamics model from rewards and uses MCTS for discrete action selection.

— **Ablations**. We consider: *(i)* our method implemented using a state predictor (h_θ being the identity function), *(ii)* our method implemented without the latent consistency loss from Equation 10, and lastly: the consistency loss replaced by either *(iii)* the *reconstruction* objective of PlaNet and Dreamer, or *(iv)* the *contrastive* objective of EfficientZero.

See Appendix G for further discussion on baselines.

Table 1. **Learning from pixels.** Return of our method (TD-MPC) and state-of-the-art algorithms on the image-based DMControl 100k benchmark used in Srinivas et al. (2020); Kostrikov et al. (2020); Ye et al. (2021). Baselines are tuned specifically for image-based RL, whereas our method is not. Results for SAC, CURL, DrQ, and PlaNet are partially obtained from Srinivas et al. (2020); Kostrikov et al. (2020), and results for Dreamer, MuZero, and EfficientZero are obtained from Hafner et al. (2020b); Ye et al. (2021). Mean and std. deviation over 10 runs. *: MuZero and EfficientZero use a discretized action space, and EfficientZero performs an additional 20k gradient steps before evaluation, whereas other methods do not. Due to dimensionality explosion under discretization, MuZero and EfficientZero cannot feasibly solve tasks with higher-dimensional action spaces, e.g., Walker Walk and Cheetah Run ($\mathcal{A} \in \mathbb{R}^6$), while our method can.

	Model-free				Model-based				Ours
100k env. steps	SAC State	SAC Pixels	CURL	DrQ	PlaNet	Dreamer	MuZero*	Eff.Zero*	**TD-MPC**
Cartpole Swingup	812±45	419±40	597±170	**759±92**	563±73	326±27	219±122	**813±19**	**770±70**
Reacher Easy	919±123	145±30	517±113	601±213	82±174	314±155	493±145	**952±34**	628±105
Cup Catch	957±26	312±63	772±241	**913±53**	710±217	246±174	542±270	**942±17**	**933±24**
Finger Spin	672±76	166±128	779±108	**901±104**	560±77	341±70	–	–	**943±59**
Walker Walk	604±317	42±12	344±132	**612±164**	221±43	277±12	–	–	**577±208**
Cheetah Run	228±95	103±38	**307±48**	**344±67**	165±123	235±137	–	–	222±88

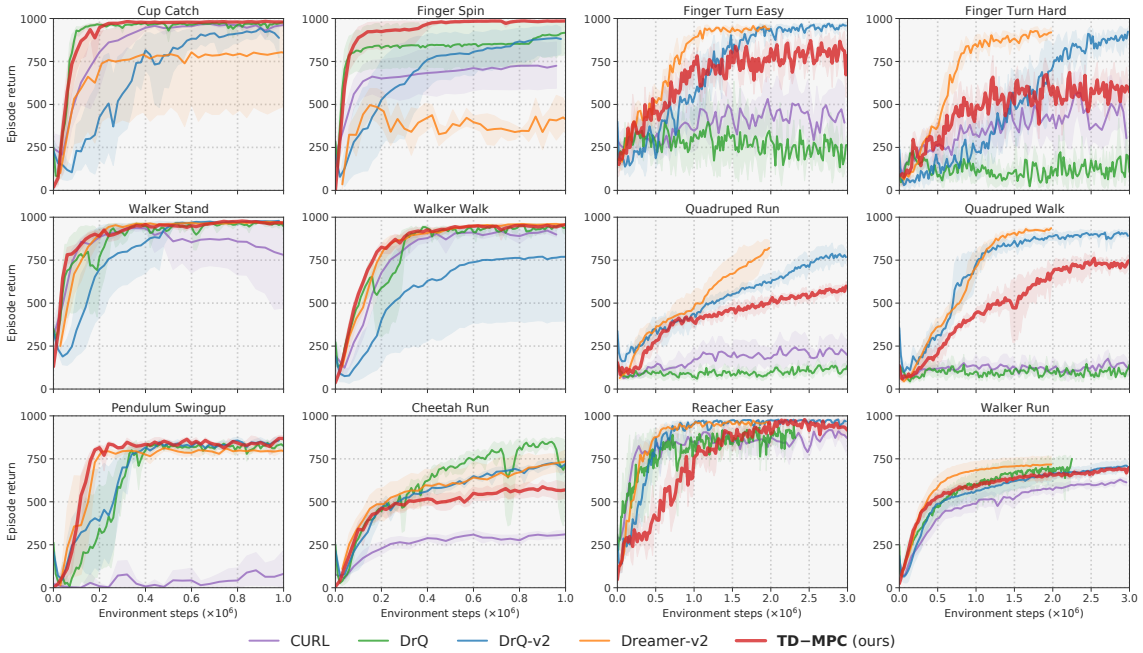

Figure 4. **Learning from pixels.** Return of our method (TD-MPC) and state-of-the-art algorithms on 12 challenging image-based DMControl tasks. We follow prior work (Hafner et al., 2020b;a; Yarats et al., 2021) and use an action repeat of 2 for all tasks. Compared to the DMControl 100k benchmark shown in Table 1, we here consider more difficult tasks with up to 30× more data. Results for DrQ-v2 and Dreamer-v2 are obtained from Yarats et al. (2021); Hafner et al. (2020a), results for DrQ are partially obtained from Kostrikov et al. (2020), and results for CURL are reproduced using their publicly available implementation (Srinivas et al., 2020). While baselines use task-dependent hyperparameters, TD-MPC uses the **same** hyperparameters for **all** tasks. Mean of 5 runs; shaded areas are 95% confidence intervals. TD-MPC consistently outperforms CURL and DrQ, and is competitive with DrQ-v2 and Dreamer-v2.

Tasks. We consider the following **92** tasks:

— **6** challenging Humanoid ($\mathcal{A} \in \mathbb{R}^{21}$) and Dog ($\mathcal{A} \in \mathbb{R}^{38}$) locomotion tasks with high-dimensional state and action spaces. Results are shown in Figure 1.

— **15** diverse continuous control tasks from DMControl, 6 of which have sparse rewards. Results shown in Figure 3.

— **6** image-based tasks from the data-efficient DMControl 100k benchmark. Results are shown in Table 1.

— **12** image-based tasks from the DMControl *Dreamer* benchmark (3M environment steps). Results in Figure 4.

— **2** multi-modal (proprioceptive data + egocentric camera) 3D locomotion tasks in which a quadruped agent navigates around obstacles. Results are shown in Figure 5 (middle).

— **50** goal-conditioned manipulation tasks from Meta-World, as well as a multi-task setting where 10 tasks are learned simultaneously. Results are shown in Figure 5 (top).

Throughout, we benchmark performance on relatively few environment steps, e.g., 3M steps for Humanoid tasks whereas prior work typically runs for 30M steps (10×).

Comparison to other methods. We find our method to outperform or match baselines in most tasks considered, generally with larger gains on complex tasks such as *Humanoid*, *Dog* (DMControl), and *Bin Picking* (Meta-World), and we note that TD-MPC is in fact ***the first documented result*** solving the complex *Dog* tasks of DMControl. Performance of LOOP is similar to SAC, and MPC with a simulator (*MPC:sim*) performs well on locomotion tasks but fails in tasks with sparse rewards. Although we did not tune our method specifically for image-based RL, we obtain results competitive with state-of-the-art model-based and model-free algorithms that are both carefully tuned for image-based RL and contain up to $15\times$ more learnable parameters. Notably, while EfficientZero produces strong results on tasks with low-dimensional action spaces, its Monte-Carlo Tree Search (MCTS) requires discretization of action spaces, which is unfeasible in high dimensions. In contrast, TD-MPC scales remarkably well to the 38-dimensional continuous action space of *Dog* tasks. Lastly, we observe inferior sample efficiency compared to SAC and LOOP on the hard exploration task *Finger Turn Hard* in Figure 3, which suggests that incorporating more sophisticated exploration strategies might be promising for future research. We defer experiments that ablate the choice of regularization loss to Appendix D, but find our proposed latent state consistency loss to yield the most consistent results.

Multi-task RL, multi-modal RL, and generalization. A common argument in favor of general-purpose models is that they can benefit from data-sharing across tasks. Therefore, we seek to answer the following question: does TOLD similarly benefit from synergies between tasks, despite its reward-centric objective? We test this hypothesis through two experiments: training a single policy to perform 10 different tasks simultaneously (Meta-World MT10), and evaluating model generalization when trained on one task (*Walk*) and transferring to a different task from the same domain (*Run*). Multi-task results are shown in Figure 5 (top), and transfer results are deferred to Appendix A. We find our method to benefit from data sharing in both experiments, and our transfer results indicate that h_θ generalizes well to new tasks, while d_θ encodes more task-specific behavior. We conjecture that, while TOLD only learns features that are predictive of reward, similar tasks often have similar reward structures, which enables sharing of information between tasks. However, we still expect general-purpose models to benefit more from *unrelated* tasks in the same environment than TOLD. Lastly, an added benefit of our task-centric objective is that it is agnostic to the input modality. To demonstrate this, we solve two multi-modal (proprioceptive data + egocentric camera) locomotion tasks using TD-MPC; results in Figure 5 (bottom). We find that TD-MPC successfully fuses information from the two input modalities, and solves the tasks. In contrast, a blind agent that does *not* have access to the egocentric camera fails. See Appendix J for further details on the multi-modal experiments, and Appendix I for details on the multi-task experiments.

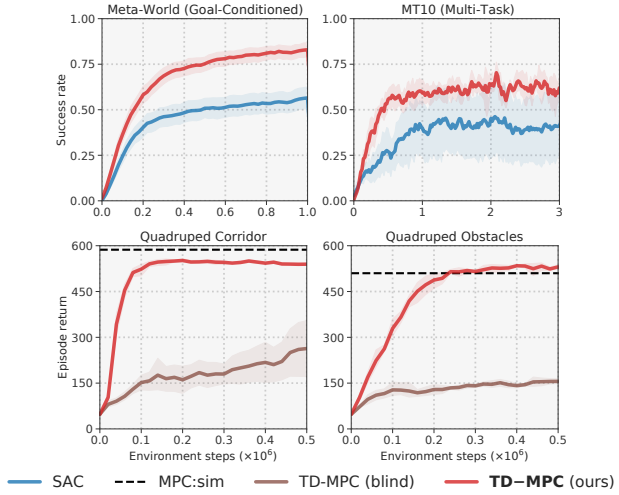

Figure 5. (top) **Meta-World.** Success rate on **50** goal-conditioned Meta-World tasks using individual policies, and a multi-task policy trained on 10 tasks simultaneously (Meta-World MT10). Individual task results shown in Appendix I. *(bottom)* **Multi-modal RL.** Episode return of TD-MPC on two multi-modal locomotion tasks using proprioceptive data + an egocentric camera. *Blind* uses only proprioceptive data. See Appendix L for visualizations. All results are means of 5 runs; shaded areas are 95% confidence intervals.

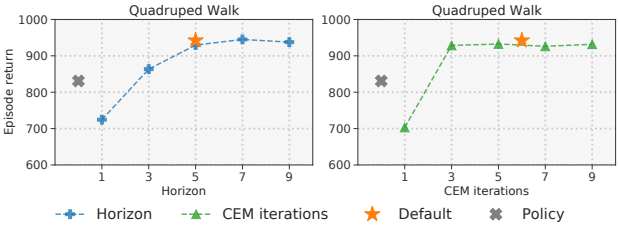

Figure 6. **Variable computational budget.** Return of TD-MPC on *Quadruped Walk* under a variable budget. We evaluate performance of fully trained agents when varying *(left)* planning horizon; *(right)* number of iterations during planning. When varying one hyperparameter, the other is fixed to the default value. We include evaluation of the learned policy π_θ, and the default setting of 6 iterations and a horizon of 5 used in training. Mean of 5 runs.

Performance vs. computational budget. We investigate the relationship between computational budget (i.e., planning horizon and number of iterations) and performance in DMControl tasks; see Figure 6. We find that, for complex tasks such as *Quadruped Walk* ($\mathcal{A} \in \mathbb{R}^{12}$), more planning generally leads to better performance. However, we also observe that we can reduce the planning cost during inference by $\mathbf{50}\%$ (compared to during training) *without* a drop in performance by reducing the number of iterations. For particularly fast inference, one can discard planning altogether and simply use the jointly learned policy π_θ; however, π_θ generally performs worse than planning. See Appendix C for additional results.

Table 2. **Wall-time.** *(top)* time to solve, and *(bottom)* time per 500k environment steps (in hours) for the *Walker Walk* and *Humanoid Stand* tasks from DMControl. We consider the tasks solved when a method achieves an average return of 940 and 800, respectively. TD-MPC solves Walker Walk $\mathbf{16}\times$ faster than LOOP while using $\mathbf{3.3}\times$ less compute per 500k steps. Mean of 5 runs.

	Walker Walk				Humanoid Stand	
Wall-time (h)	SAC	LOOP	MPC:sim	**TD-MPC**	SAC	**TD-MPC**
time to solve ↓	0.41	7.72	0.91	0.47	9.31	9.39
h/500k steps ↓	1.41	18.5	–	5.60	1.82	12.94

Training wall-time. To better ground our results, we report the training wall-time of TD-MPC compared to SAC, LOOP that is most similar to our method, and MPC with a ground-truth simulator (non-parametric). Methods are benchmarked on a single RTX3090 GPU. Results are shown in Table 2. TD-MPC solves *Walker Walk* $\mathbf{16}\times$ faster than LOOP and matches the time-to-solve of SAC on both *Walker Walk* and *Humanoid Stand* while being significantly more sample efficient. Thus, our method effectively closes the time-to-solve gap between model-free and model-based methods. This is a nontrivial reduction, as LOOP is already known to be, e.g., $12\times$ faster than the purely model-based method, POLO (Lowrey et al., 2019; Sikchi et al., 2022). We provide additional experiments on inference times in Appendix H.

6. Related Work

Temporal Difference Learning. Popular model-free off-policy algorithms such as DDPG (Lillicrap et al., 2016) and SAC (Haarnoja et al., 2018) represent advances in deep TD-learning based on a large body of literature (Sutton, 1988; Mnih et al., 2013; Hasselt et al., 2016; Mnih et al., 2016; Fujimoto et al., 2018; Kalashnikov et al., 2018; Espeholt et al., 2018; Pourchot & Sigaud, 2019; Kalashnikov et al., 2021). Both DDPG and SAC learn a policy π_θ and value function Q_θ, but do not learn a model. Kalashnikov et al. (2018); Shao et al. (2020); Kalashnikov et al. (2021) also learn Q_θ, but replace or augment π_θ with model-free CEM. Instead, we jointly learn a model, value function, and policy using TD-learning, and interact using sampling-based planning.

Model-based RL. A common paradigm is to learn a model of the environment that can be used for planning (Ebert et al., 2018; Zhang et al., 2018; Janner et al., 2019; Hafner et al., 2019; Lowrey et al., 2019; Kaiser et al., 2020; Bhardwaj et al., 2020; Yu et al., 2020; Schrittwieser et al., 2020; Nguyen et al., 2021) or for training a model-free algorithm with generated data (Pong et al., 2018; Ha & Schmidhuber, 2018; Hafner et al., 2020b; Sekar et al., 2020). For example, Zhang et al. (2018); Ha & Schmidhuber (2018); Hafner et al. (2019; 2020b) learn a dynamics model using a video prediction loss, Yu et al. (2020); Kidambi et al. (2020) consider model-based RL in the offline setting, and MuZero/EfficientZero (Schrittwieser et al., 2020; Ye et al., 2021) learn a latent dynamics model using reward prediction. EfficientZero is most similar to ours in terms of model learning, but its MCTS-based action selection is inherently incompatible with continuous action spaces. Finally, while learning a terminal value function for MPC has previously been proposed (Negenborn et al., 2005; Lowrey et al., 2019; Bhardwaj et al., 2020; Hatch & Boots, 2021), we are (to the best of our knowledge) the first to jointly learn model and value function through TD-learning in continuous control.

Hybrid algorithms. Several prior works aim to develop algorithms that combine model-free and model-based elements (Nagabandi et al., 2018; Buckman et al., 2018; Pong et al., 2018; Hafez et al., 2019; Sikchi et al., 2022; Wang & Ba, 2020; Clavera et al., 2020; Hansen et al., 2021; Morgan et al., 2021; Bhardwaj et al., 2021; Margolis et al., 2021), many of which are orthogonal to our contributions. For example, Clavera et al. (2020) and Buckman et al. (2018); Lowrey et al. (2019) use a learned model to improve policy and value learning, respectively, through generated trajectories. LOOP (Sikchi et al., 2022) extends SAC with a learned state prediction model and constrains planned trajectories to be close to those of SAC, whereas we *replace* the parameterized policy by planning with TD-MPC and learn a *task-oriented* latent dynamics model.

We provide a qualitative comparison of key components in TD-MPC and prior work in Appendix B.

7. Conclusions and Future Directions

We are excited that our TD-MPC framework, despite being markedly distinct from previous work in the way that the model is learned and used, is already able to outperform model-based and model-free methods on diverse continuous control tasks, and (with trivial modifications) simultaneously match state-of-the-art on image-based RL tasks. Yet, we believe that there is ample opportunity for performance improvements by extending the TD-MPC framework. For example, by using the learned model in creative ways (Clavera et al., 2020; Buckman et al., 2018; Lowrey et al., 2019), incorporating better exploration strategies, or improving the model through architectural innovations.

Acknowledgements

This project is supported, in part, by grants from NSF CCF-2112665 (TILOS), and gifts from Meta, Qualcomm.

The authors would like to thank Yueh-Hua Wu, Ruihan Yang, Sander Tonkens, Tongzhou Mu, and Yuzhe Qin for helpful discussions.

References

Agarwal, R., Schwarzer, M., Castro, P. S., Courville, A., and Bellemare, M. G. Deep reinforcement learning at the edge of the statistical precipice. *Advances in Neural Information Processing Systems*, 2021.

Argenson, A. and Dulac-Arnold, G. Model-based offline planning. *ArXiv*, abs/2008.05556, 2021.

Bhardwaj, M., Handa, A., Fox, D., and Boots, B. Information theoretic model predictive q-learning. *ArXiv*, abs/2001.02153, 2020.

Bhardwaj, M., Choudhury, S., and Boots, B. Blending mpc & value function approximation for efficient reinforcement learning. *ArXiv*, abs/2012.05909, 2021.

Buckman, J., Hafner, D., Tucker, G., Brevdo, E., and Lee, H. Sample-efficient reinforcement learning with stochastic ensemble value expansion. In *NeurIPS*, 2018.

Chen, X. and He, K. Exploring simple siamese representation learning. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 15745–15753, 2021.

Chua, K., Calandra, R., McAllister, R., and Levine, S. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *NeurIPS*, 2018.

Clavera, I., Fu, Y., and Abbeel, P. Model-augmented actor-critic: Backpropagating through paths. *ArXiv*, abs/2005.08068, 2020.

Ebert, F., Finn, C., Dasari, S., Xie, A., Lee, A. X., and Levine, S. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *ArXiv*, abs/1812.00568, 2018.

Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., Legg, S., and Kavukcuoglu, K. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. *ArXiv*, abs/1802.01561, 2018.

Fujimoto, S., Hoof, H. V., and Meger, D. Addressing function approximation error in actor-critic methods. *ArXiv*, abs/1802.09477, 2018.

Ha, D. and Schmidhuber, J. Recurrent world models facilitate policy evolution. In *Advances in Neural Information Processing Systems 31*, pp. 2451–2463. Curran Associates, Inc., 2018.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., and Levine, S. Soft actor-critic algorithms and applications. *ArXiv*, abs/1812.05905, 2018.

Hafez, M. B., Weber, C., Kerzel, M., and Wermter, S. Curious meta-controller: Adaptive alternation between model-based and model-free control in deep reinforcement learning. *2019 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2019.

Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., and Davidson, J. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, pp. 2555–2565, 2019.

Hafner, D., Lillicrap, T., Norouzi, M., and Ba, J. Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*, 2020a.

Hafner, D., Lillicrap, T. P., Ba, J., and Norouzi, M. Dream to control: Learning behaviors by latent imagination. *ArXiv*, abs/1912.01603, 2020b.

Hansen, N. and Wang, X. Generalization in reinforcement learning by soft data augmentation. In *International Conference on Robotics and Automation (ICRA)*, 2021.

Hansen, N., Jangir, R., Sun, Y., Alenyà, G., Abbeel, P., Efros, A. A., Pinto, L., and Wang, X. Self-supervised policy adaptation during deployment. In *International Conference on Learning Representations (ICLR)*, 2021.

Hasselt, H. V., Guez, A., and Silver, D. Deep reinforcement learning with double q-learning. In *Aaai*, 2016.

Hatch, N. and Boots, B. The value of planning for infinite-horizon model predictive control. *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7372–7378, 2021.

Janner, M., Fu, J., Zhang, M., and Levine, S. When to trust your model: Model-based policy optimization. *ArXiv*, abs/1906.08253, 2019.

Kaiser, L., Babaeizadeh, M., Milos, P., Osinski, B., Campbell, R. H., Czechowski, K., Erhan, D., Finn, C., Kozakowski, P., Levine, S., Sepassi, R., Tucker, G., and Michalewski, H. Model-based reinforcement learning for atari. *ArXiv*, abs/1903.00374, 2020.

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

Kalashnikov, D., Varley, J., Chebotar, Y., Swanson, B., Jonschkowski, R., Finn, C., Levine, S., and Hausman, K. Mt-opt: Continuous multi-task robotic reinforcement learning at scale. *ArXiv*, abs/2104.08212, 2021.

Kidambi, R., Rajeswaran, A., Netrapalli, P., and Joachims, T. Morel : Model-based offline reinforcement learning. *ArXiv*, abs/2005.05951, 2020.

Kostrikov, I., Yarats, D., and Fergus, R. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. *International Conference on Learning Representations*, 2020.

Lillicrap, T., Hunt, J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971, 2016.

Lowrey, K., Rajeswaran, A., Kakade, S. M., Todorov, E., and Mordatch, I. Plan online, learn offline: Efficient learning and exploration via model-based control. *ArXiv*, abs/1811.01848, 2019.

Margolis, G., Chen, T., Paigwar, K., Fu, X., Kim, D., Kim, S., and Agrawal, P. Learning to jump from pixels. In *CoRL*, 2021.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In *ICML*, 2016.

Morgan, A. S., Nandha, D., Chalvatzaki, G., D'Eramo, C., Dollar, A. M., and Peters, J. Model predictive actor-critic: Accelerating robot skill acquisition with deep reinforcement learning. *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6672–6678, 2021.

Nagabandi, A., Kahn, G., Fearing, R. S., and Levine, S. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7559–7566, 2018.

Negenborn, R. R., De Schutter, B., Wiering, M. A., and Hellendoorn, H. Learning-based model predictive control for markov decision processes. *IFAC Proceedings Volumes*, 38(1):354–359, 2005. 16th IFAC World Congress.

Nguyen, T. D., Shu, R., Pham, T., Bui, H. H., and Ermon, S. Temporal predictive coding for model-based planning in latent space. In *ICML*, 2021.

Pong, V. H., Gu, S. S., Dalal, M., and Levine, S. Temporal difference models: Model-free deep rl for model-based control. *ArXiv*, abs/1802.09081, 2018.

Pourchot, A. and Sigaud, O. Cem-rl: Combining evolutionary and gradient-based methods for policy search. *ArXiv*, abs/1810.01222, 2019.

Rubinstein, R. Y. Optimization of computer simulation models with rare events. *European Journal of Operational Research*, 99:89–112, 1997.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. Prioritized experience replay. *CoRR*, abs/1511.05952, 2016.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., Lillicrap, T. P., and Silver, D. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588 7839:604–609, 2020.

Sekar, R., Rybkin, O., Daniilidis, K., Abbeel, P., Hafner, D., and Pathak, D. Planning to explore via self-supervised world models. *ArXiv*, abs/2005.05960, 2020.

Shao, L., You, Y., Yan, M., Sun, Q., and Bohg, J. Grac: Self-guided and self-regularized actor-critic. *arXiv preprint arXiv:2009.08973*, 2020.

Sikchi, H., Zhou, W., and Held, D. Learning off-policy with online planning. In *Conference on Robot Learning*, pp. 1622–1633. PMLR, 2022.

Srinivas, A., Laskin, M., and Abbeel, P. Curl: Contrastive unsupervised representations for reinforcement learning. *arXiv preprint arXiv:2004.04136*, 2020.

Sutton, R. Learning to predict by the method of temporal differences. *Machine Learning*, 3:9–44, 08 1988. doi: 10.1007/BF00115009.

Sutton, R. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 2005.

Tassa, Y., Erez, T., and Todorov, E. Synthesis and stabilization of complex behaviors through online trajectory optimization. *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4906–4913, 2012.

Tassa, Y., Doron, Y., Muldal, A., Erez, T., Li, Y., de Las Casas, D., Budden, D., Abdolmaleki, A., et al. Deepmind control suite. Technical report, DeepMind, 2018.

Wang, T. and Ba, J. Exploring model-based planning with policy networks. *ArXiv*, abs/1906.08649, 2020.

Williams, G., Aldrich, A., and Theodorou, E. A. Model predictive path integral control using covariance variable importance sampling. *ArXiv*, abs/1509.01149, 2015.

Yarats, D. and Kostrikov, I. Soft actor-critic (sac) implementation in pytorch. https://github.com/denisyarats/pytorch_sac, 2020.

Yarats, D., Fergus, R., Lazaric, A., and Pinto, L. Mastering visual continuous control: Improved data-augmented reinforcement learning. *arXiv preprint arXiv:2107.09645*, 2021.

Ye, W., Liu, S., Kurutach, T., Abbeel, P., and Gao, Y. Mastering atari games with limited data. *ArXiv*, abs/2111.00210, 2021.

Yu, T., Quillen, D., He, Z., Julian, R., Hausman, K., Finn, C., and Levine, S. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning (CoRL)*, 2019.

Yu, T., Thomas, G., Yu, L., Ermon, S., Zou, J. Y., Levine, S., Finn, C., and Ma, T. Mopo: Model-based offline policy optimization. *ArXiv*, abs/2005.13239, 2020.

Zhang, M., Vikram, S., Smith, L., Abbeel, P., Johnson, M. J., and Levine, S. Solar: Deep structured latent representations for model-based reinforcement learning. *ArXiv*, abs/1808.09105, 2018.

Ziebart, B. D., Maas, A., Bagnell, J. A., and Dey, A. K. Maximum entropy inverse reinforcement learning. In *Proceedings of the 23rd National Conference on Artificial Intelligence*, volume 3, 2008.

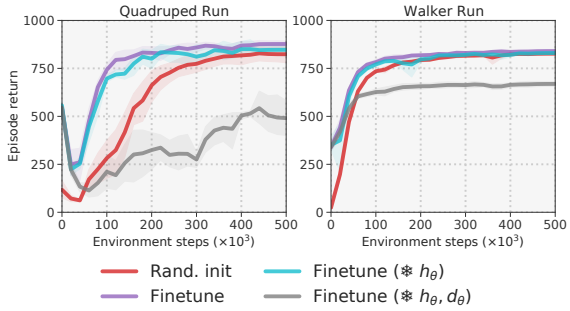

Figure 7. **Model generalization.** Return of our method under three different settings: *(Rand. init)* TD-MPC trained from scratch on the two *Run* tasks; *(Finetune)* TD-MPC initially trained on *Walk* tasks and then finetuned online on *Run* tasks without any weights frozen; *(Finetune, freeze h_θ)* same setting as before, but with the encoder h_θ frozen; and *(Finetune, freeze h_θ, d_θ)* both encoder h_θ and latent dynamics predictor d_θ frozen. Mean of 5 runs; shaded areas are 95% confidence intervals.

A. Model Generalization

We investigate the transferability of a TOLD model between related tasks. Specifically, we consider model transfer in two locomotion domains, *Walker* and *Quadruped*, where we first train policies on *Walk* tasks and then finetune the learned model on *Run* tasks. We finetune in an online setting, i.e., the only difference between training from scratch and finetuning is the weight initialization, and we keep all hyperparameters identical. Results from the experiment are shown in Figure 7. When finetuning the full TOLD model, we find our method to converge considerably faster, suggesting that TOLD does indeed learn features that transfer between related tasks. We perform two additional finetuning experiments: freezing parameters of the representation h_θ, and freezing parameters of both h_θ *and* the latent dynamics predictor d_θ during finetuning. We find that freezing h_θ nearly matches our results for finetuning without frozen weights, indicating that h_θ learns to encode information that transfers between tasks. However, when finetuning with both h_θ, d_θ frozen, rate of convergence degrades substantially, which suggests that d_θ tends to encode more task-specific behavior.

B. Comparison to Prior Work

We here extend our discussion of related work in Section 6. Table 3 provides a qualitative comparison of key components of TD-MPC and prior model-based and model-free approaches, e.g., comparing model objectives, use of a (terminal) value function, and inference-time behavior. While different aspects of TD-MPC have been explored in prior work, we are the first to propose a complete framework for MPC with a model learned by TD-learning.

C. Variable Computational Budget

This section supplements our experiments in Figure 6 on a variable computational budget for planning during inference; additional results are shown in Figure 8. We observe that the gap between planning performance and policy performance tends to be larger for tasks with high-dimensional action spaces such as the two *Quadruped* tasks. We similarly find that performance varies relatively little when the computational budget is changed for tasks with simple dynamics (e.g., *Cartpole* tasks) compared to tasks with more complex dynamics. We find that our default hyperparameters ($H = 5$ and 6 iterations; shown as a star in Figure 8) strikes a good balance between compute and performance.

D. Latent Dynamics Objective

We ablate the choice of latent dynamics objective by replacing our proposed latent state consistency loss in Equation 10 with *(i)* a contrastive loss similar to that of Ye et al. (2021); Hansen & Wang (2021), and *(ii)* a reconstruction objective similar to that of Ha & Schmidhuber (2018); Hafner et al. (2019; 2020b). Specifically, for (i) we adopt the recently proposed SimSiam (Chen & He, 2021) self-supervised framework and implement the projection layer as an MLP with 2 hidden layers and output size 32, and the predictor head is an MLP with 1 hidden layer. All layers use ELU activations and a hidden size of 256. Consistent with the public implementations of Ye et al. (2021); Hansen & Wang (2021),

Table 3. **Comparison to prior work.** We compare key components of **TD-MPC** to prior model-based and model-free approaches. *Model objective* describes which objective is used to learn a (latent) dynamics model, *value* denotes whether a value function is learned, *inference* provides a simplified view of action selection at inference time, *continuous* denotes whether an algorithm supports continuous action spaces, and *compute* is a holistic estimate of the relative computational cost of methods during training and inference. We use *policy w/ CEM* to indicate inference based primarily on a learned policy, and vice-versa.

Method	Model objective	Value	Inference	Continuous	Compute
SAC	✗	✓	Policy	✓	Low
QT-Opt	✗	✓	CEM	✓	Low
MPC:sim	Ground-truth model	✗	CEM	✓	High
POLO	Ground-truth model	✓	CEM	✓	High
LOOP	State prediction	✓	Policy w/ CEM	✓	Moderate
PlaNet	Image prediction	✗	CEM	✓	High
Dreamer	Image prediction	✓	Policy	✓	Moderate
MuZero	Reward/value pred.	✓	MCTS w/ policy	✗	Moderate
EfficientZero	Reward/value pred. + contrast.	✓	MCTS w/ policy	✗	Moderate
TD-MPC (ours)	Reward/value pred. + latent pred.	✓	CEM w/ policy	✓	Low

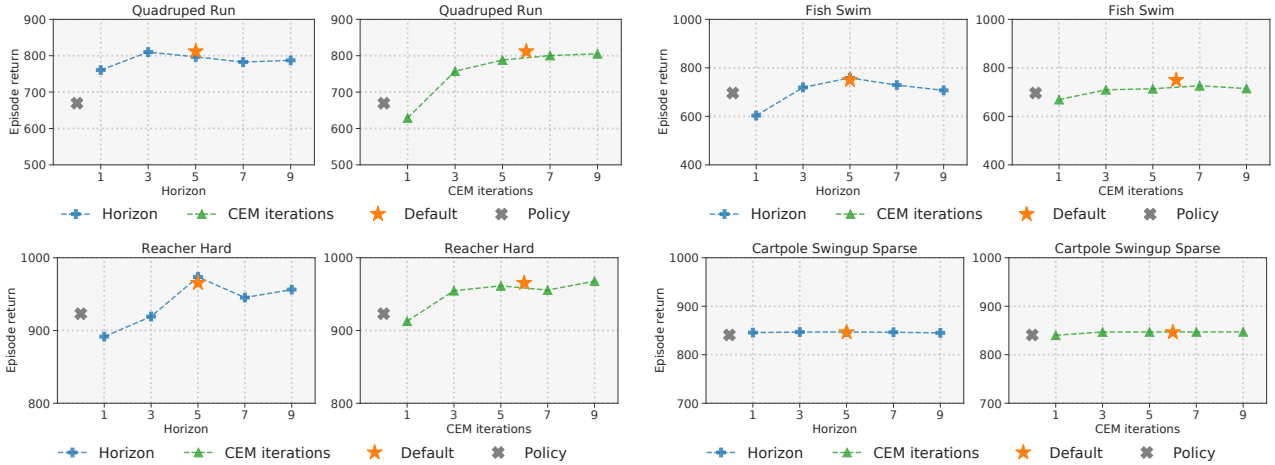

Figure 8. **Variable computational budget.** Return of our method (TD-MPC) under a variable computational budget. In addition to the task in Figure 6, we provide results on four other tasks from DMControl: *Quadruped Run* ($\mathcal{A} \in \mathbb{R}^{12}$), *Fish Swim* ($\mathcal{A} \in \mathbb{R}^{5}$), *Reacher Hard* ($\mathcal{A} \in \mathbb{R}^{2}$), and *Cartpole Swingup Sparse* ($\mathcal{A} \in \mathbb{R}$). We evaluate performance of fully trained agents when varying *(blue)* planning horizon; *(green)* number of iterations during planning. For completeness, we also include evaluation of the jointly learned policy π_θ, as well as the default setting of 6 iterations and a horizon of 5 used during training. Higher values require more compute. Mean of 5 runs.

we find it beneficial to apply BatchNorm in the projection and predictor modules. We also find that using a higher loss coefficient of $c_3 = 100$ (up from 2) produces slightly better results. For (ii) we implement the decoder for state reconstruction by mirroring the encoder; an MLP with 1 hidden layer and ELU activations. We also include a *no regularization* baseline for completeness. Results are shown in Figure 10.

E. Exploration by planning

We investigate the role that planning by TD-MPC has in exploration. Figure 9 shows the average std. deviation of our planning procedure after the final iteration of planning for the three Humanoid tasks: Stand, Walk, and Run, listed in order of increasing difficulty. We observe that the std. deviation (and thus degree of exploration) is decreasing as training progresses, and converges as the task becomes solved. Generally, we find that exploration decreases slower for hard tasks, which we conjecture is due to larger variance in reward and value estimates. As such, the TD-MPC framework inherently balances exploration and exploitation.

F. Implementation Details

We provide an overview of the implementation details of our method in Section 5. For completeness, we list all relevant hyperparameters in Table 4. As discussed in Appendix G, we adopt most hyperparameters from the SAC implementation (Yarats & Kostrikov, 2020). Following previous work (Hafner et al., 2019), we use a task-specific action repeat hyperparameter for DMControl that is constant across all

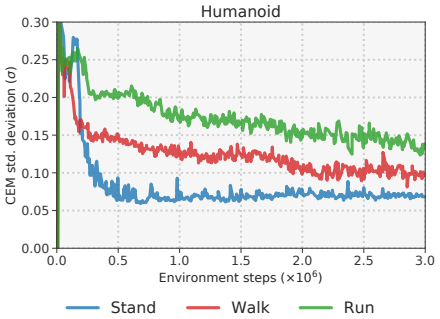

Figure 9. **Exploration by planning.** Average std. deviation (σ) of our planning procedure after the final iteration of planning over the course of training. Results are shown for the three Humanoid tasks: Stand, Walk, and Run, listed in order of increasing difficulty.

methods; see Table 7 for a list of values. For state-based experiments, we implement the representation function h_θ using an MLP with a single hidden layer of dimension 256. For image-based experiments, h_θ is a 4-layer CNN with kernel sizes $(7, 5, 3, 3)$, stride $(2, 2, 2, 2)$, and 32 filters per layer. All other components are implemented using 2-layer MLPs with dimension 512. Following prior work (Yarats & Kostrikov, 2020; Srinivas et al., 2020; Kostrikov et al., 2020), we apply layer normalization to the value function. Weights and biases in the last layer of the reward predictor R_θ and value function Q_θ are zero-initialized to reduce model and value biases in the early stages of training, and all other fully-connected layers use orthogonal initialization; the SAC and LOOP baselines are implemented similarly. We do not find it consistently better to use larger networks neither for state-based nor image-based experiments. In multi-task experiments, we augment the state input with a one-hot task vector. In multi-modal experiments, we encode state and image separately and sum the features. We provide a PyTorch-like summary of our task-oriented latent dynamics model in the following. For clarity, we use *S*, *Z*, and *A* to denote the dimensionality of states, latent states, and actions, respectively, and report the total number of learnable parameters for our TOLD model initialized for the *Walker Run* task ($\mathcal{S} \in \mathbb{R}^{24},\ \mathcal{A} \in \mathbb{A}^{6}$).

```
Total parameters: approx. 1,507,000
(h): Sequential(
  (0): Linear(in_features=S, out_features=256)
  (1): ELU(alpha=1.0)
  (2): Linear(in_features=256, out_features=Z))
(d): Sequential(
  (0): Linear(in_features=Z+A, out_features=512)
  (1): ELU(alpha=1.0)
  (2): Linear(in_features=512, out_features=512)
  (3): ELU(alpha=1.0)
  (4): Linear(in_features=512, out_features=Z))
(R): Sequential(
  (0): Linear(in_features=Z+A, out_features=512)
  (1): ELU(alpha=1.0)
  (2): Linear(in_features=512, out_features=512)
  (3): ELU(alpha=1.0)
  (4): Linear(in_features=512, out_features=1))
(pi): Sequential(
  (0): Linear(in_features=Z, out_features=512)
  (1): ELU(alpha=1.0)
  (2): Linear(in_features=512, out_features=512)
  (3): ELU(alpha=1.0)
  (4): Linear(in_features=512, out_features=A))
(Q1): Sequential(
  (0): Linear(in_features=Z+A, out_features=512)
  (1): LayerNorm((512,), elementwise_affine=True)
  (2): Tanh()
  (3): Linear(in_features=512, out_features=512)
  (4): ELU(alpha=1.0)
  (5): Linear(in_features=512, out_features=1))
(Q2): Sequential(
  (0): Linear(in_features=Z+A, out_features=512)
  (1): LayerNorm((512,), elementwise_affine=True)
  (2): Tanh()
  (3): Linear(in_features=512, out_features=512)
  (4): ELU(alpha=1.0)
  (5): Linear(in_features=512, out_features=1))
```

Additionally, PyTorch-like pseudo-code for training our TOLD model (codified version of Algorithm 2) is shown below:

```
def update(replay_buffer):
  """
  A single gradient update of our TOLD model.
  h, R, Q, d: TOLD components.
  c1, c2, c3: loss coefficients.
  rho: temporal loss coefficient.
  """
  states, actions, rewards = replay_buffer.sample()

  # Encode first observation
  z = h(states[0])

  # Recurrently make predictions
  reward_loss = 0
  value_loss = 0
  consistency_loss = 0
  for t in range(H):
    r = R(z, actions[t])
    q1, q2 = Q(z, actions[t])
    z = d(z, actions[t])

    # Compute targets and losses
    z_target = h_target(states[t+1])
    td_target = compute_td(rewards[t], states[t+1])
    reward_loss += rho**t * mse(r, rewards[t])
    value_loss += rho**t * \
      (mse(q1, td_target) + mse(q2, td_target))
    consistency_loss += rho**t * mse(z, z_target)

  # Update
  total_loss = c1 * reward_loss + \
               c2 * value_loss + \
               c3 * consistency_loss
  total_loss.backward()
  optim.step()

  # Update slow-moving average
  update_target_network()
```

G. Extended Description of Baselines

We tune the performance of both our method and baselines to perform well on DMControl and then subsequently benchmark algorithms on Meta-World using the same choice of hyperparameters. Below, we provide additional details on our efforts to tune the baseline implementations.

SAC. We adopt the implementation of Yarats & Kostrikov (2020) which has been used extensively in the literature as a

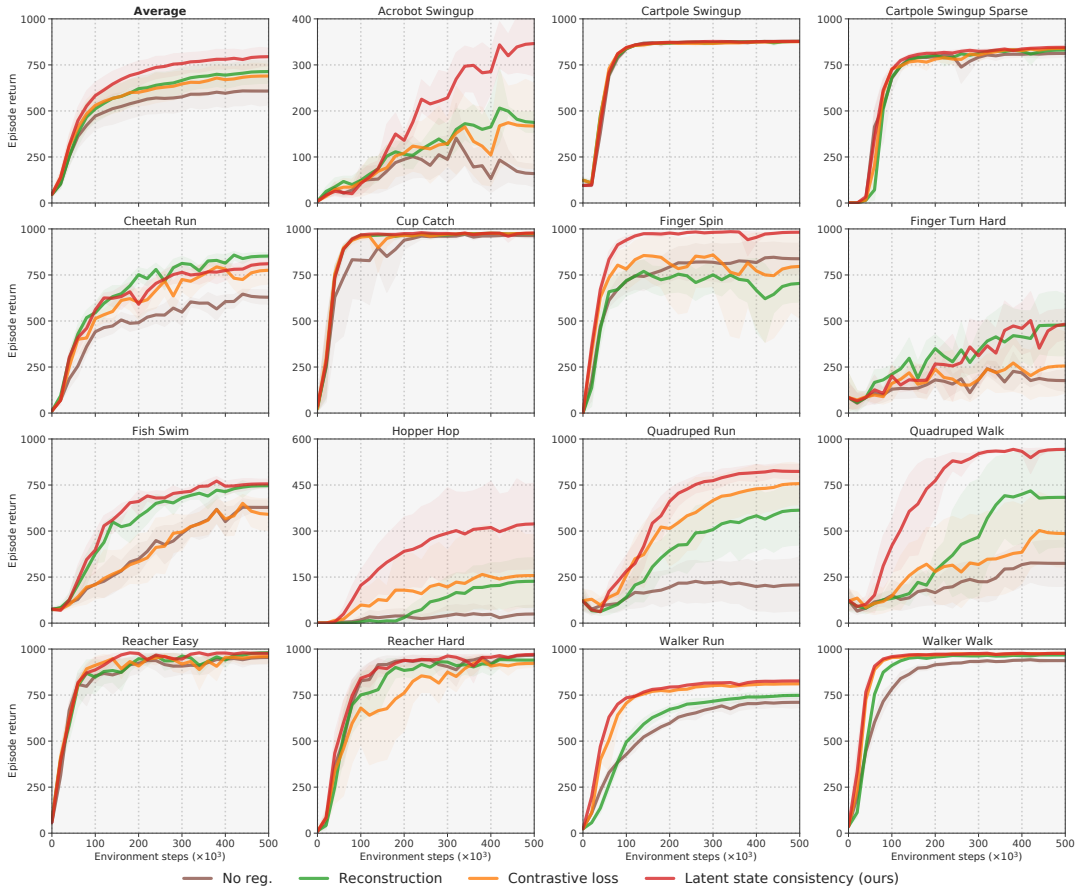

Figure 10. **Latent dynamics objective.** Return of our method (TD-MPC) using different latent dynamics objectives in addition to reward and value prediction. 15 state-based continuous control tasks from DMControl (Tassa et al., 2018). *No reg.* uses no regularization term, *reconstruction* uses a state prediction loss, *contrastive loss* adopts the contrastive objective of Ye et al. (2021); Hansen & Wang (2021), and *latent state consistency* corresponds to Equation 10. Mean of 5 runs; shaded areas are 95% confidence intervals. In the top left, we visualize results averaged across all 15 tasks. Both reconstruction and contrastive losses improve over the baseline without regularization, but our proposed latent state consistency loss yields more consistent results.

benchmark implementation for state-based DMControl. We use original hyperparameters except for the target network momentum coefficient ζ, where we find it beneficial for both SAC, LOOP, and our method to use a faster update of $\zeta = 0.99$ as opposed to 0.995 in the original implementation. Additionally, we decrease the batch size from 1024 to 512 for fair comparison to our method. For completeness, we list important hyperparameters for the SAC baseline in Table 5.

LOOP. We benchmark against the official implementation from Sikchi et al. (2022), but note that LOOP has – to the best of our knowledge – not previously been benchmarked on DMControl nor Meta-World. Therefore, we do our best to adapt its hyperparameters. As in the SAC implementation, we find LOOP to perform better using $\zeta = 0.99$ than its original value of 0.995, and we increase the batch size from 256 to 512. Lastly, we set the number of seed steps to $1,000$ (down from $10,000$) to match the SAC implementation. As LOOP uses SAC as backbone learning algorithm, we found these changes to be beneficial. LOOP-specific hyperparameters are listed in Table 6.

MPC:sim. We compare TD-MPC to a vanilla MPC algorithm using a ground-truth model of the environment (simulator), but no terminal value function. As such, this baseline is non-parametric. We use the same MPC implementation as in our method (MPPI; Williams et al. (2015)). As planning with a simulator is computationally intensive, we limit the planning horizon to 10 (which is still $2\times$ as much as TD-MPC), and we reduce the number of iterations to 4 (our method uses 6), as we find MPC to converge faster when using the ground-truth model. At each iteration, we sample $N = 200$ trajectories and update distribution parameters using the top-20 (10%) sampled action sequences. We keep all other hyperparameters consistent with our method. Because of the limited planning horizon, this MPC baseline generally performs well for locomotion tasks where local solutions are sufficient, but tends to fail at tasks with, for example, sparse rewards.

Table 4. **TD-MPC hyperparameters.** We here list hyperparameters for TD-MPC with TOLD and emphasize that we use the same parameters for SAC whenever possible.

Hyperparameter	Value
Discount factor (γ)	0.99
Seed steps	$5,000$
Replay buffer size	Unlimited
Sampling technique	PER ($\alpha = 0.6, \beta = 0.4$)
Planning horizon (H)	5
Initial parameters (μ^0, σ^0)	$(0, 2)$
Population size	512
Elite fraction	64
Iterations	12 (Humanoid) 8 (Dog, pixels) 6 (otherwise)
Policy fraction	5%
Number of particles	1
Momentum coefficient	0.1
Temperature (τ)	0.5
MLP hidden size	512
MLP activation	ELU
Latent dimension	100 (Humanoid, Dog) 50 (otherwise)
Learning rate	3e-4 (Dog, pixels) 1e-3 (otherwise)
Optimizer (θ)	Adam ($\beta_1 = 0.9, \beta_2 = 0.999$)
Temporal coefficient (λ)	0.5
Reward loss coefficient (c_1)	0.5
Value loss coefficient (c_2)	0.1
Consistency loss coefficient (c_3)	2
Exploration schedule (ϵ)	$0.5 \rightarrow 0.05$ (25k steps)
Planning horizon schedule	$1 \rightarrow 5$ (25k steps)
Batch size	2048 (Dog) 256 (pixels) 512 (otherwise)
Momentum coefficient (ζ)	0.99
Steps per gradient update	1
θ^- update frequency	2

Table 5. **SAC hyperparameters.** We list the most important hyperparameters for the SAC baseline. Note that we mostly follow the implementation of Yarats & Kostrikov (2020) but improve upon certain hyperparameter choices, e.g., the momentum coefficient ζ and values specific to the Dog tasks.

Hyperparameter	Value
Discount factor (γ)	0.99
Seed steps	$1,000$
Replay buffer size	Unlimited
Sampling technique	Uniform
MLP hidden size	1024
MLP activation	RELU
Latent dimension	100 (Humanoid, Dog) 50 (otherwise)
Optimizer (θ)	Adam ($\beta_1 = 0.9, \beta_2 = 0.999$)
Optimizer (α of SAC)	Adam ($\beta_1 = 0.5, \beta_2 = 0.999$)
Learning rate (θ)	3e-4 (Dog) 1e-3 (otherwise)
Learning rate (α of SAC)	1e-4
Batch size	2048 (Dog) 512 (otherwise)
Momentum coefficient (ζ)	0.99
Steps per gradient update	1
θ^- update frequency	2

Table 6. **LOOP hyperparameters.** We list general SAC hyperparameters shared by LOOP in Table 5, and list only hyperparameters specific to LOOP here. We use the official implementation from Sikchi et al. (2022) but list its hyperparameters for completeness. Note that we – as in the SAC implementation – use a different batch size and momentum coefficient than in Sikchi et al. (2022), as we find this to marginally improve performance on DMControl.

Hyperparameter	Value
Planning horizon (H)	3
Population size	100
Elite fraction	20%
Iterations	5
Policy fraction	5%
Number of particles	4
Momentum coefficient	0.1
MLP hidden size	256
MLP activation	ELU/RELU
Ensemble size	5

No latent ablation. We make the following change to our method: replacing h_θ with the identity function, i.e., $\mathbf{x} = h_\theta(\mathbf{x})$. As such, environment dynamics are modelled by forward prediction directly in the state space, with the consistency loss effectively degraded to a state prediction loss. This ablation makes our method more similar to prior work on model-based RL from states (Janner et al., 2019; Lowrey et al., 2019; Sikchi et al., 2022; Argenson & Dulac-Arnold, 2021). However, unlike previous work that decouples model learning from policy and value learning, we still back-propagate gradients from the reward and value objectives through the model, which is a stronger baseline.

No consistency regularization. We set the coefficient c_3 corresponding to the latent state consistency loss in Equation 10 to 0, such that the TOLD model is trained only with the reward and value prediction losses. This ablation makes our method more similar to MuZero (Schrittwieser et al., 2020).

Other baselines. Results for other baselines are obtained from related work. Specifically, results for SAC, CURL, DrQ, and PlaNet are obtained from Srinivas et al. (2020) and Kostrikov et al. (2020), and results for Dreamer, MuZero, and EfficientZero are obtained from Hafner et al. (2020b) and Ye et al. (2021).

H. Inference Time

In the experiments of Section 5, we investigate the relationship between performance and the computational budget of planning with TD-MPC. For completeness, we also evaluate the relationship between computational budget and inference time. Figure 11 shows the inference time of TD-MPC as the planning horizon and number of iterations is varied. As in previous experiments, we benchmark inference times on a single RTX3090 GPU. Unsurprisingly, we

Table 7. **Action repeat.** We adopt action repeat hyperparameters for DMControl from previous work (Hafner et al., 2019; Kostrikov et al., 2020) for state-based experiments as well as the DMControl 100k benchmark; we list all values below. For the DMControl *Dreamer* benchmark, all methods use an action repeat of 2 regardless of the task. We do not use action repeat for Meta-World.

Task	Action repeat
Humanoid	2
Dog	2
Walker	2
Finger	2
Cartpole	8
Other (DMControl)	4
Meta-World	1

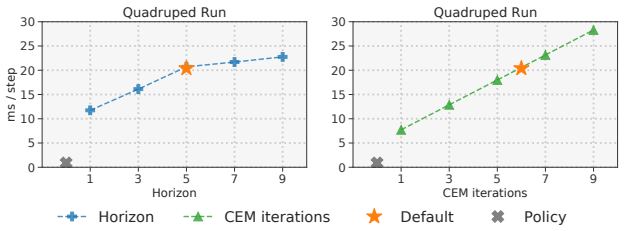

Figure 11. **Inference time under a variable budget.** Milliseconds per decision step for TD-MPC on the *Quadruped Run* task under a variable computational budget. We evaluate performance of fully trained agents when varying *(left)* planning horizon; *(right)* number of iterations during planning. When varying one hyperparameter, the other is fixed to the default value. For completeness, we also include the inference time of the learned policy π_θ, and the default setting of 6 iterations and a horizon of 5 used during training.

Table 8. **Meta-World MT10.** As our performance metric reported in Figure 5 differs from that of the Meta-World *v2* benchmark proposal (Yu et al., 2019), we here report results for our SAC baseline using the same *maximum per-task success rate* metric used for the MT10 multi-task experiment from the original paper.

Task	Max. success rate
Window Close	1.00
Window Open	1.00
Door Open	1.00
Peg Insert Side	0.00
Drawer Open	0.85
Pick Place	0.00
Reach	1.00
Button Press Down	1.00
Push	0.00
Drawer Close	1.00

find that there is an approximately linear relationship between computational budget and inference time. However, it is worth noting that our default settings used during training only require approximately 20ms per step, i.e., 50Hz, which is fast enough for many real-time robotics applications such as manipulation, navigation, and to some extent locomotion (assuming an on-board GPU). For applications where inference time is critical, the computational budget can be adjusted to meet requirements. For example, we found in Figure 8 that we can reduce the planning horizon of TD-MPC on the *Quadruped Run* task from 5 to 1 with no significant reduction in performance, which reduces inference time to approximately 12ms per step. While the performance of the model-free policy learned jointly with TD-MPC indeed is lower than that of planning, it is however still nearly $6\times$ faster than planning at inference time.

I. Meta-World

We provide learning curves and success rates for individual Meta-World (Yu et al., 2019) tasks in Figure 14. Due to the sheer number of tasks, we choose to only visualize the first 24 tasks (sorted alphabetically) out of the total of 50 tasks from Meta-World. Note that we use Meta-World *v2* and that we consider the goal-conditioned versions of the tasks, which are considered harder than the single-goal variant often used in related work. We generally find that SAC is competitive to TD-MPC in most tasks, but that TD-MPC is far more sample efficient in tasks that involve complex manipulation, e.g., *Bin Picking*, *Box Close*, and *Hammer*. Successful trajectories for each of these three tasks are visualized in Figure 15. Generally, we choose to focus on sample-efficiency for which we empirically find 1M environment steps (3M for multi-task experiments) to be sufficient for achieving non-trivial success rates in Meta-World. As the original paper reports *maximum per-task success rate* for multi-task experiments rather than average success rate, we also report this metric for our SAC baseline in Table 8. We find that our SAC baseline is strikingly competitive with the original paper results considering that we evaluate over just 3M steps.

J. Multi-Modal RL

We demonstrate the ability of TD-MPC to successfully fuse information from multiple input modalities (proprioceptive data + an egocentric camera) in two 3D locomotion tasks:

— **Quadruped Corridor**, where the agent needs to move along a corridor with constant target velocity. To succeed, the agent must perceive the corridor walls and adjust its walking direction accordingly.

— **Quadruped Obstacles**, where the agent needs to move along a corridor filled with obstacles that obstruct vision and

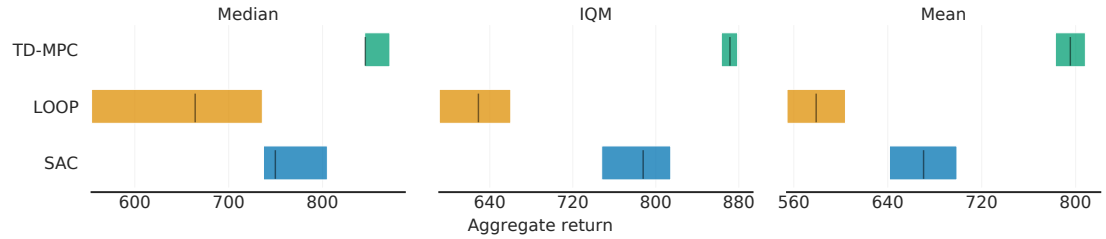

Figure 12. **Rliable metrics.** Median, interquantile median (IQM), and mean performance of TD-MPC and baselines on the 15 state-based DMControl tasks. Confidence intervals are estimated using the percentile bootstrap with stratified sampling, per recommendation of Agarwal et al. (2021). Higher values are better. 5 seeds.

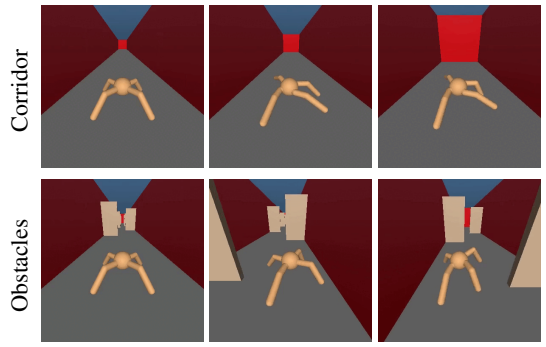

Figure 13. **Multi-modal RL.** Visualization of the two multi-modal 3D locomotion tasks that we construct.

forces the agent to move in a zig-zag pattern with constant target velocity. To succeed, the agent must perceive both the corridor walls and obstacles, and continuously adjust its walking direction.

Trajectories from the two tasks are visualized in Figure 13.

K. Additional Metrics

We report additional (aggregate) performance metrics of SAC, LOOP, and TD-MPC on the set of 15 state-based DMControl tasks using the *rliable* toolkit provided by Agarwal et al. (2021). Concretely, we report the aggregate median, interquantile mean (IQM), and mean returns with 95% confidence intervals based on the episode returns of trained (after 500k environment steps) agents. As recommended by Agarwal et al. (2021), confidence intervals are estimated using the percentile bootstrap with stratified sampling.

L. Task Visualizations

Figure 15 provides visualizations of successful trajectories generated by TD-MPC on seven tasks from DMControl and Meta-World, all of which TD-MPC solves in less than 1M environment steps. In all seven trajectories, we display only key frames in the trajectory, as actual episode lengths are 1000 (DMControl) and 500 (Meta-World). For full video trajectories, refer to `https://nicklashansen.github.io/td-mpc`.

Additional material on the following pages ↓

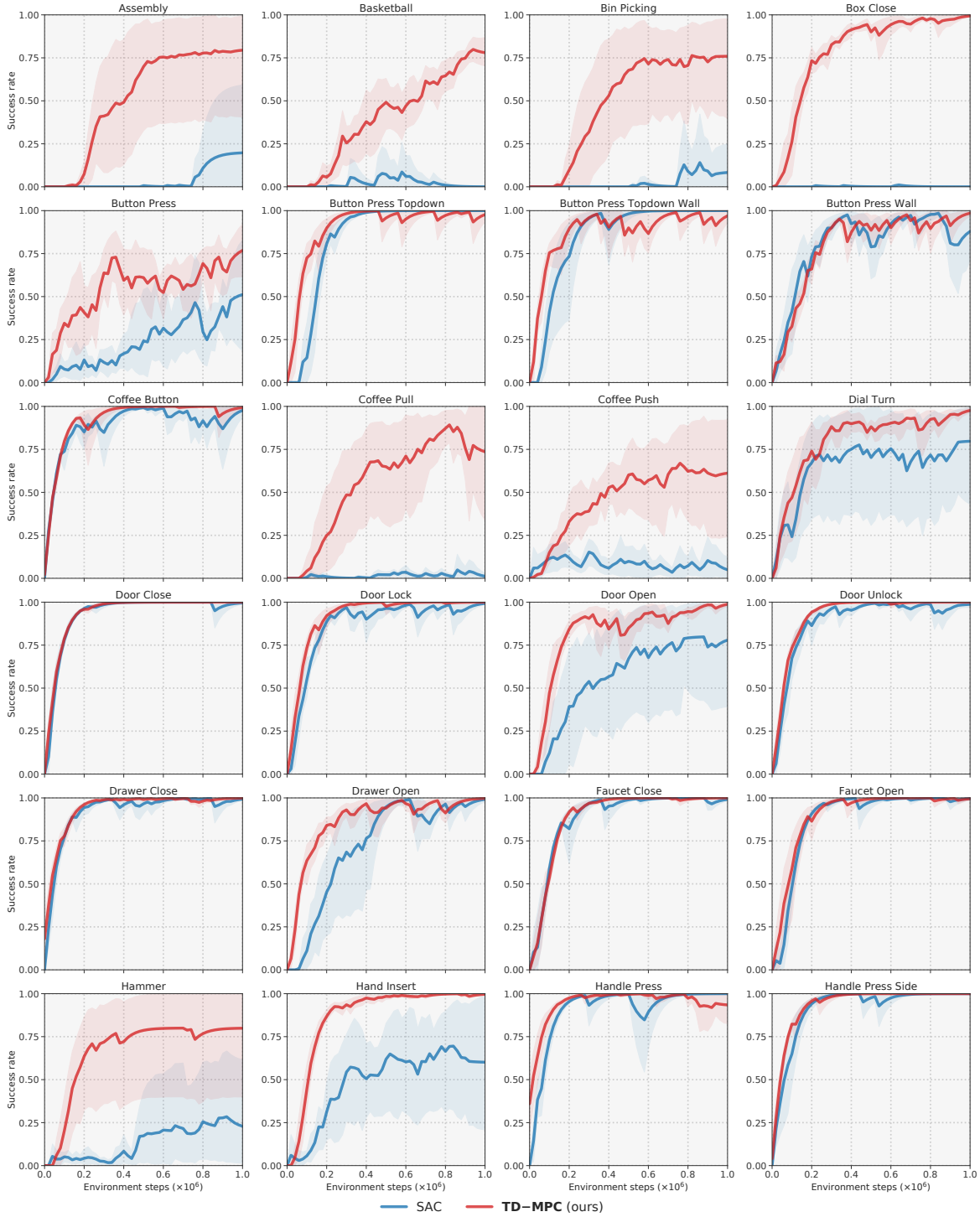

Figure 14. **Individual Meta-World tasks.** Success rate of our method (TD-MPC) and SAC on diverse manipulation tasks from Meta-World (Yu et al., 2019). We use the goal-conditioned version of Meta-World, which is considered harder than the fixed-goal version. Due to the large number of tasks (50), we choose to visualize only the first 24 tasks (sorted alphabetically). Mean of 5 runs; shaded areas are 95% confidence intervals. Our method is capable of solving complex tasks (e.g., *Basketball*) where SAC achieves a relatively small success rate. Note that we use Meta-World *v2* and performances are therefore not comparable to previous work using *v1*.

Figure 15. **Visualizations.** We visualize trajectories generated by our method on seven selected tasks from the two benchmarks, listed (from top to bottom) as follows: *(1)* Dog Walk, a challenging locomotion task that has a high-dimensional action space ($\mathcal{A} \in \mathbb{R}^{38}$); *(2)* Humanoid Walk, a challenging locomotion task ($\mathcal{A} \in \mathbb{R}^{21}$); *(3)* Quadruped Run, a four-legged locomotion task ($\mathcal{A} \in \mathbb{R}^{12}$); *(4)* Finger Turn Hard, a hard exploration task with sparse rewards; *(5)* Bin Picking, a 3-d pick-and-place task; *(6)* Box Close, a 3-d manipulation task; and lastly *(7)* Hammer, another 3d-manipulation task. In all seven trajectories, we display only key frames in the trajectory. Actual episode lengths are 1000 (DMControl) and 500 (Meta-World). Our method (TD-MPC) is capable of solving each of these tasks in less than 1M environment steps. Video results are available at `https://nicklashansen.github.io/td-mpc`.