Temporal Difference Learning for Model Predictive Control

Nicklas Hansen ¹  Xiaolong Wang ²  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 ad-

[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. Copyright 2022 by the author(s).

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.

vantages 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.

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
TD-Learning for MPC

(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 Temporal Difference Learning for Model Predictive Control (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 $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma, p_0 \rangle$, where $\mathcal{S} \in \mathbb{R}^n$ and $\mathcal{A} \in \mathbb{R}^m$ are continuous state and action spaces, $\mathcal{T}: \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}_+$ is the transition (dynamics) function, $\mathcal{R}: \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ 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 $\Pi_\theta: \mathcal{S} \mapsto \mathcal{A}$ with parameters $\theta$ such that discounted return $\mathbb{E}_{r_t \sim \Pi_\theta}[\sum_{i=1}^{\infty} \gamma^i r_i]$, $r_t \sim \mathcal{R}(\cdot|s_t, a_t)$ is maximized along a trajectory $\Gamma = (s_0, a_0, s_1, a_1, \ldots)$ following $\Pi_\theta$ by sampling an action $a_t \sim \Pi_\theta(\cdot|s_t)$ and reaching state $s_{t+1} \sim \mathcal{T}(\cdot|s_t, a_t)$ at each decision step $t$.

Fitted Q-iteration. Model-free TD-learning algorithms aim to estimate an optimal state-action value function $Q^*: \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ using a parametric value function $Q_\theta(s, a) = \mathbb{E}[\mathcal{R}(s, a) + \gamma Q^*(s', a')]$ over $s'$, $a'$ is the state and action at the following step, and $\theta$ 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 $Q_\theta$ using the update rule

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

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

Model Predictive Control. In actor-critic RL algorithms, $\Pi$ is typically a policy parameterized by a neural network that learns to approximate $\Pi_\theta(\cdot|s) \approx \arg \max_a \mathbb{E}[Q_\theta(s, a)] \forall s \in \mathcal{S}$, i.e., the globally optimal policy. In control, $\Pi$ 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 $a_{t+H}$ over a finite horizon $H$ and executing the first action $a_t$, known as Model Predictive Control (MPC):

$$
\Pi_{\theta}^{\text{MPC}}(s_t) = \arg \max_{a_{t+H}} \mathbb{E} \left[ \sum_{i=t}^{H} \gamma^i \mathcal{R}(s_i, a_i) \right],
$$

where $\gamma$, 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., $\mu, \sigma$ 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 $s_{t+H}$ and
We propose $\Omega$ where $z$ weighted average of the estimated top-trajectories, as highlighted in red in Algorithm 1. We detail with a task-oriented latent dynamics model and terminal dynamics and reward signal, respectively, a terminal state-action value function $Q$. MPPI is an MPC algorithm that iteratively updates parameters $(\mu, \sigma)$ of a time-dependent multivariate Gaussian with diagonal parameterization using neural networks as control for planning (denoted $\Pi$), learned models $d_\theta$, $R_\theta$ of the (latent) dynamics and reward signal, respectively, a terminal state-action value function $Q_\theta$, and a parameterized policy $\pi_\theta$ 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.

Algorithm 1: TD-MPC (inference)

Require: $\theta$: learned network parameters $\mu^0, \sigma^0$: initial parameters for $\mathcal{N}$ $N, N_\tau$: num sample/policy trajectories $s_t, H$: current state, rollout horizon

1: Encode state $z_t \leftarrow h_\theta(s_t)$ \hfill (Assuming TOLD model)
2: for each iteration $j = 1..J$ do
3: Sample $N$ traj. of len. $H$ from $\mathcal{N}(\mu^{j-1} - \epsilon, (\sigma^{j-1})^2 I)$
4: Sample $N_\tau$ traj. of length $H$ using $\pi_\theta, d_\theta$
5: for all $N + N_\tau$ trajectories $(s_t, a_t, \ldots, a_{t+H})$ do
6: for step $t = 0..H - 1$ do
7: $\phi_\Gamma = \phi_\Gamma + \gamma^t R_\theta(z_t, a_t)$ \hfill (Reward)
8: $z_{t+1} \leftarrow d_\theta(z_t, a_t)$ \hfill (Latent transition)
9: $\phi_\Gamma = \phi_\Gamma + \gamma^{H-t} Q_\theta(z_H, a_H)$ \hfill (Terminal value)

10: $\mu^j, \sigma^j =$ Equation 4 (and Equation 5)
11: return $a \sim \mathcal{N}(\mu^j, (\sigma^j)^2 I)$

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 $\Pi_\theta$), learned models $d_\theta$, $R_\theta$ of the (latent) dynamics and reward signal, respectively, a terminal state-action value function $Q_\theta$, and a parameterized policy $\pi_\theta$ 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 $d_\theta$, and estimate the total return $\phi_\Gamma$ of a sampled trajectory $\Gamma$ as

$$\phi_\Gamma \triangleq \mathbb{E}_\Gamma \left[ \gamma^H Q_\theta(z_H, a_H) + \sum_{t=0}^{H-1} \gamma^t R_\theta(z_t, a_t) \right], \quad (3)$$

where $z_{t+1} = d_\theta(z_t, a_t)$ and $a_t \sim \mathcal{N}(\mu^{j-1}, (\sigma^{j-1})^2 I)$ at iteration $j - 1$, as highlighted in red in Algorithm 1. We select the top-$k$ returns $\phi_\Gamma^*$ and obtain new parameters $\mu^j, \sigma^j$ at iteration $j$ from a $\phi_\Gamma^*$-normalized empirical estimate:

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

where $\Omega_i = e^{\tau(\phi_\Gamma^*)_i}$, $\tau$ is a temperature parameter controlling the “sharpness” of the weighting, and $\Gamma_i^*$ denotes the $i$th top-$k$ trajectory corresponding to return estimate $\phi_\Gamma^*$. 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 $\mu$ obtained at the previous step (Argenson & Dulac-Arnold, 2021), but always use a large initial variance to avoid local minima.

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 $\pi_\theta$ 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 $\sigma$ decays varies wildly between tasks, leading to (potentially poor) local optima for small $\sigma$. To promote consistent exploration across tasks, we constrain the std. deviation of the sampling distribution such that, for a $\mu^j$ obtained from Equation 4 at iteration $j$, we instead update $\sigma^j$ to

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

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 $\pi_\theta$ in addition to planning procedure $\Pi_\theta$, and augments the sampling procedure with additional samples from $\pi_\theta$ (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 $\phi_t$. 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 $\pi_\theta$ 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 Task-Oriented Latent Dynamics (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:

- **Representation:** $z_t = h_\theta(s_t)$
- **Latent dynamics:** $z_{t+1} = d_\theta(z_t, a_t)$
- **Reward:** $r_t = R_\theta(z_t, a_t)$ (6)
- **Value:** $\hat{q}_t = Q_\theta(z_t, a_t)$
- **Policy:** $\pi_t \sim \pi_\theta(z_t)$

Given an observation $s_t$ observed at time $t$, a representation network $h_\theta$ encodes $s_t$ into a latent representation $z_t$. From $z_t$ and an action $a_t$ taken at time $t$, TOLD then predicts (i) the latent dynamics (latent representation $z_{t+1}$ 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.

4. Task-Oriented Latent Dynamics Model

To be used in conjunction with TD-MPC, we propose a Task-Oriented Latent Dynamics (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:

- **Representation:** $z_t = h_\theta(s_t)$
- **Latent dynamics:** $z_{t+1} = d_\theta(z_t, a_t)$
- **Reward:** $r_t = R_\theta(z_t, a_t)$ (6)
- **Value:** $\hat{q}_t = Q_\theta(z_t, a_t)$
- **Policy:** $\pi_t \sim \pi_\theta(z_t)$

Figure 2. Training our TOLD model. A trajectory $\Gamma_{0:H}$ of length $H$ is sampled from a replay buffer, and the first observation $s_0$ is encoded by $h_\theta$ into a latent representation $z_0$. Then, TOLD recurrently predicts the following latent states $z_1, z_2, \ldots, z_H$, as well as a value $\hat{q}$, reward $r$, and action $a$ for each latent state, and we optimize TOLD using Equation 7. Subsequent observations are encoded using target net $h_{\theta-}(\theta)$: slow-moving average of $\theta$ 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

$$J(\theta; \Gamma) = \sum_{t=0}^{t+H} \lambda^{t-H} L(\theta; \Gamma_t),$$

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

$$L(\theta; \Gamma_t) = c_1 \| R_\theta(z_t, a_t) - r_t \|^2_2$$

$$+ c_2 \| Q_\theta(z_t, a_t) - (r_t + \gamma Q_\theta(z_{t+1}, \pi_\theta(z_{t+1})) \|_2^2$$

$$+ c_3 \| d_\theta(z_t, a_t) - h_\theta- (s_{t+1}) \|^2_2$$

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 $(z_t, a_t)$, 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 \( z_t = h_\theta(s_t), z_{t+1} = d_\theta(z_t, a_t), \ldots, z_{t+H} = d_\theta(z_{t+H-1}, a_{t+H-1}) \) such that only the first observation \( s_t \) is encoded using \( h_\theta \) 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; Janner et al., 2019; Lowrey et al., 2020b; Sikchi et al., 2022). We use an exponential moving average \( \theta^\ast \) of the online network parameters \( \theta \) for computing the value target (Lillicrap et al., 2016), and similarly also use \( \theta^\ast \) for the latent state consistency target \( h_\theta(s_{t+1}) \). The policy \( \pi_\theta \) 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 \( \max_{\pi_\theta} Q_\theta(z_t, a_t) \), which is extremely costly to compute using planning (Lowrey et al., 2019). Therefore, we instead learn a policy \( \pi_\theta \) that maximizes \( Q_\theta \) by minimizing the objective

\[
J_\pi(\theta; \Gamma) = - \sum_{i=t+H}^{t+H} \lambda^{i-t} Q_\theta(z_i, \pi_\theta(\text{sg}(z_i))),
\]

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, \( \text{sg} \) denotes the stop-grad operator, and Equation 11 is optimized only wrt. policy parameters. While we empirically observe that for complex tasks the learned \( \pi_\theta \) 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 \( z_{t+1} = d_\theta(z_t, a_t) \) at time \( t + 1 \) to be similar to the latent representation of the corresponding ground-truth observation \( h_\theta(s_{t+1}) \), 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.

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

---

**Algorithm 2** TOLD (training)

| Require: \( \theta, \theta^\ast \): randomly initialized network parameters \( \eta, \tau, \lambda, B \): learning rate, coefficients, buffer |
|:-------------|-----|-----|-----|
| 1: while not tired do |
| 2: // Collect episode with TD-MPC from \( s_0 \sim p_0 \): |
| 3: \( \text{for step } t = 0 \ldots T \text{ do} \) |
| 4: \( a_t \sim \Pi_\theta(\cdot | h_\theta(s_t)) \) \hspace{1cm} \( \triangleright \) Sample with TD-MPC |
| 5: \( (s_{t+1}, r_t) \sim \mathcal{T}(\cdot | s_t, a_t) \) \hspace{1cm} \( \triangleright \) Step env. |
| 6: \( B \leftarrow B \cup (s_t, a_t, r_t, s_{t+1}) \) \hspace{1cm} \( \triangleright \) Add to buffer |
| 7: // Update TOLD using collected data in B: |
| 8: \( \text{for num updates per episode do} \) |
| 9: \( \{s_t, a_t, r_t, s_{t+1}\}_{t+H} \sim B \) \hspace{1cm} \( \triangleright \) Sample traj. |
| 10: \( z_t = h_\theta(s_t) \) \hspace{1cm} \( \triangleright \) Encode first observation |
| 11: \( J = 0 \) \hspace{1cm} \( \triangleright \) Initialize J for loss accumulation |
| 12: \( \text{for } i = t \ldots t + H \text{ do} \) |
| 13: \( \hat{r}_i = R_\theta(z_i, a_i) \) \hspace{1cm} \( \triangleright \) Equation 8 |
| 14: \( \hat{q}_i = Q_\theta(z_i, a_i) \) \hspace{1cm} \( \triangleright \) Equation 9 |
| 15: \( z_{i+1} = d_\theta(z_i, a_i) \) \hspace{1cm} \( \triangleright \) Equation 10 |
| 16: \( \hat{a}_i = \pi_\theta(z_i) \) \hspace{1cm} \( \triangleright \) Equation 11 |
| 17: \( J \leftarrow J + \lambda^{i-t} \mathcal{L}(z_{i+1}, \hat{r}_i, \hat{q}_i, \hat{a}_i) \) \hspace{1cm} \( \triangleright \) Equation 7 |
| 18: \( \theta \leftarrow \theta - \frac{\eta}{J} \nabla_J J \) \hspace{1cm} \( \triangleright \) Update online network |
| 19: \( \theta^\ast \leftarrow (1 - \tau) \theta^\ast + \tau \theta \) \hspace{1cm} \( \triangleright \) Update target network |

**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](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 \( \epsilon \) of \( \Pi_\theta \) and \( \pi_\theta \) 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 \( \pi_\theta \)), and we compute \( \mu, \sigma \) 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.

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× 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 (hθ 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 (A ∈ ℝ^6), while our method can.

<table>
<thead>
<tr>
<th>100k env. steps</th>
<th>SAC State</th>
<th>SAC Pixels</th>
<th>CURL</th>
<th>DrQ</th>
<th>PlaNet</th>
<th>Dreamer</th>
<th>MuZero*</th>
<th>Eff.Zero*</th>
<th>TD-MPC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cartpole Swingup</td>
<td>812±45</td>
<td>419±40</td>
<td>597±170</td>
<td>759±92</td>
<td>563±73</td>
<td>326±27</td>
<td>219±122</td>
<td>813±19</td>
<td>770±70</td>
</tr>
<tr>
<td>Reacher Easy</td>
<td>919±123</td>
<td>145±30</td>
<td>517±113</td>
<td>601±213</td>
<td>82±174</td>
<td>314±155</td>
<td>493±145</td>
<td>952±34</td>
<td>628±105</td>
</tr>
<tr>
<td>Cup Catch</td>
<td>957±26</td>
<td>312±63</td>
<td>772±241</td>
<td>913±58</td>
<td>710±217</td>
<td>246±174</td>
<td>542±270</td>
<td>942±17</td>
<td>933±24</td>
</tr>
<tr>
<td>Finger Spin</td>
<td>672±76</td>
<td>166±128</td>
<td>779±108</td>
<td>901±104</td>
<td>560±77</td>
<td>341±70</td>
<td>–</td>
<td>–</td>
<td>943±59</td>
</tr>
<tr>
<td>Walker Walk</td>
<td>604±117</td>
<td>42±12</td>
<td>344±132</td>
<td>612±164</td>
<td>221±43</td>
<td>277±112</td>
<td>–</td>
<td>–</td>
<td>577±208</td>
</tr>
<tr>
<td>Cheetah Run</td>
<td>228±95</td>
<td>103±38</td>
<td>307±48</td>
<td>344±67</td>
<td>165±123</td>
<td>235±137</td>
<td>–</td>
<td>–</td>
<td>222±88</td>
</tr>
</tbody>
</table>

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 (A ∈ ℝ^{21}) and Dog (A ∈ 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 MetaWorld, 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× 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 \( h_0 \) generalizes well to new tasks, while \( d_0 \) 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.

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 (\( A \in \mathbb{R}^{12} \)), more planning generally leads to better performance. However, we also observe that we can reduce the planning cost during inference by 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 \( \pi_\theta \); however, \( \pi_\theta \) 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 $16 \times$ faster than LOOP while using $3.3 \times$ less compute per 500k steps. Mean of 5 runs.

<table>
<thead>
<tr>
<th></th>
<th>Walker Walk</th>
<th>Humanoid Stand</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Wall-time</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(h) SAC</td>
<td>(h) SAC</td>
</tr>
<tr>
<td>time to solve</td>
<td>0.41</td>
<td>0.91</td>
</tr>
<tr>
<td>h/500k steps</td>
<td>1.41</td>
<td>5.60</td>
</tr>
<tr>
<td>loop</td>
<td>7.72</td>
<td>18.5</td>
</tr>
<tr>
<td>TD-MPC</td>
<td>0.91</td>
<td>5.60</td>
</tr>
<tr>
<td>SAC</td>
<td>9.31</td>
<td>18.5</td>
</tr>
<tr>
<td>TD-MPC</td>
<td>9.39</td>
<td>12.94</td>
</tr>
</tbody>
</table>

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 $16 \times$ 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 $\pi_{\theta}$ and value function $Q_{\phi}$, but do not learn a model. Kalashnikov et al. (2018); Shao et al. (2020); Kalashnikov et al. (2021) also learn $Q_{\phi}$, but replace or augment $\pi_{\theta}$ 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; Bhrawaj 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., 2020; 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; Bhrawaj 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; Bhrawaj 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


---

**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 $h_\theta$)* same setting as before, but with the encoder $h_\theta$ frozen; and *(Finetune, freeze $h_\theta$, $d_\theta$)* both encoder $h_\theta$ and latent dynamics predictor $d_\theta$ 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 $h_\theta$, and freezing parameters of both $h_\theta$ and the latent dynamics predictor $d_\theta$ during finetuning. We find that freezing $h_\theta$ nearly matches our results for finetuning without frozen weights, indicating that $h_\theta$ learns to encode information that transfers between tasks. However, when finetuning with both $h_\theta, d_\theta$ frozen, rate of convergence degrades substantially, which suggests that $d_\theta$ 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.

<table>
<thead>
<tr>
<th>Method</th>
<th>Model objective</th>
<th>Value</th>
<th>Inference</th>
<th>Continuous</th>
<th>Compute</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAC</td>
<td>x</td>
<td>✓</td>
<td>Policy</td>
<td>✓</td>
<td>Low</td>
</tr>
<tr>
<td>QF-Opt</td>
<td>x</td>
<td>✓</td>
<td>CEM</td>
<td>✓</td>
<td>Low</td>
</tr>
<tr>
<td>MPC:sim</td>
<td>Ground-truth model</td>
<td>x</td>
<td>CEM</td>
<td>✓</td>
<td>High</td>
</tr>
<tr>
<td>POLO</td>
<td>Ground-truth model</td>
<td>✓</td>
<td>CEM</td>
<td>✓</td>
<td>High</td>
</tr>
<tr>
<td>LOOP</td>
<td>State prediction</td>
<td>✓</td>
<td>Policy w/ CEM</td>
<td>✓</td>
<td>Moderate</td>
</tr>
<tr>
<td>PlaNet</td>
<td>Image prediction</td>
<td>x</td>
<td>CEM</td>
<td>✓</td>
<td>High</td>
</tr>
<tr>
<td>Dreamer</td>
<td>Image prediction</td>
<td>✓</td>
<td>Policy</td>
<td>✓</td>
<td>Moderate</td>
</tr>
<tr>
<td>MuZero</td>
<td>Reward/value pred.</td>
<td>✓</td>
<td>MCTS w/ policy</td>
<td>✓</td>
<td>Moderate</td>
</tr>
<tr>
<td>EfficientZero</td>
<td>Reward/value pred. + contrast.</td>
<td>✓</td>
<td>MCTS w/ policy</td>
<td>✓</td>
<td>Moderate</td>
</tr>
<tr>
<td>TD-MPC (ours)</td>
<td>Reward/value pred. + latent pred.</td>
<td>✓</td>
<td>CEM w/ policy</td>
<td>✓</td>
<td>Low</td>
</tr>
</tbody>
</table>

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 \((A \in \mathbb{R}^{12})\), Fish Swim \((A \in \mathbb{R}^{5})\), Reacher Hard \((A \in \mathbb{R}^{2})\), and Cartpole Swingup Sparse \((A \in \mathbb{R})\). We evaluate performance of fully trained agents when varying \(\text{(blue)}\) planning horizon; \(\text{(green)}\) number of iterations during planning. For completeness, we also include evaluation of the jointly learned policy \(\pi_\theta\), 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
methods; see Table 7 for a list of values. For state-based experiments, we implement the representation function $h_\theta$ using an MLP with a single hidden layer of dimension 256. For image-based experiments, $h_\theta$ 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 $R_\theta$ and value function $Q_\theta$ 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 ($S \in \mathbb{R}^{24}, A \in \mathbb{A}^6$).

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=512)
(3): ELU(alpha=1.0)
(4): Linear(in_features=512, out_features=1))

(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=1))

(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=1))

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
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× 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.
We here list 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.

### Table 6. LOOP hyperparameters

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

**No latent ablation.** We make the following change to our method: replacing \( h_0 \) with the identity function, i.e., \( x = h_0(x) \). 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).

### 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.

<table>
<thead>
<tr>
<th>Task</th>
<th>Action repeat</th>
</tr>
</thead>
<tbody>
<tr>
<td>Humanoid</td>
<td>2</td>
</tr>
<tr>
<td>Dog</td>
<td>2</td>
</tr>
<tr>
<td>Walker</td>
<td>2</td>
</tr>
<tr>
<td>Finger</td>
<td>2</td>
</tr>
<tr>
<td>Cartpole</td>
<td>8</td>
</tr>
<tr>
<td>Other (DMControl)</td>
<td>4</td>
</tr>
<tr>
<td>Meta-World</td>
<td>1</td>
</tr>
</tbody>
</table>

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 $\pi_\theta$, 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.

<table>
<thead>
<tr>
<th>Task</th>
<th>Max. success rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Window Close</td>
<td>1.00</td>
</tr>
<tr>
<td>Window Open</td>
<td>1.00</td>
</tr>
<tr>
<td>Door Open</td>
<td>1.00</td>
</tr>
<tr>
<td>Peg Insert Side</td>
<td>0.00</td>
</tr>
<tr>
<td>Drawer Open</td>
<td>0.85</td>
</tr>
<tr>
<td>Pick Place</td>
<td>0.00</td>
</tr>
<tr>
<td>Reach</td>
<td>1.00</td>
</tr>
<tr>
<td>Button Press Down</td>
<td>1.00</td>
</tr>
<tr>
<td>Push</td>
<td>0.00</td>
</tr>
<tr>
<td>Drawer Close</td>
<td>1.00</td>
</tr>
</tbody>
</table>

We 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× 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.

Additional material on the following pages ↓

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](https://nicklashansen.github.io/td-mpc).
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.
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 \( A \in \mathbb{R}^{38} \); (2) Humanoid Walk, a challenging locomotion task \( (A \in \mathbb{R}^{21}) \); (3) Quadruped Run, a four-legged locomotion task \( (A \in \mathbb{R}^{12}) \); (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.