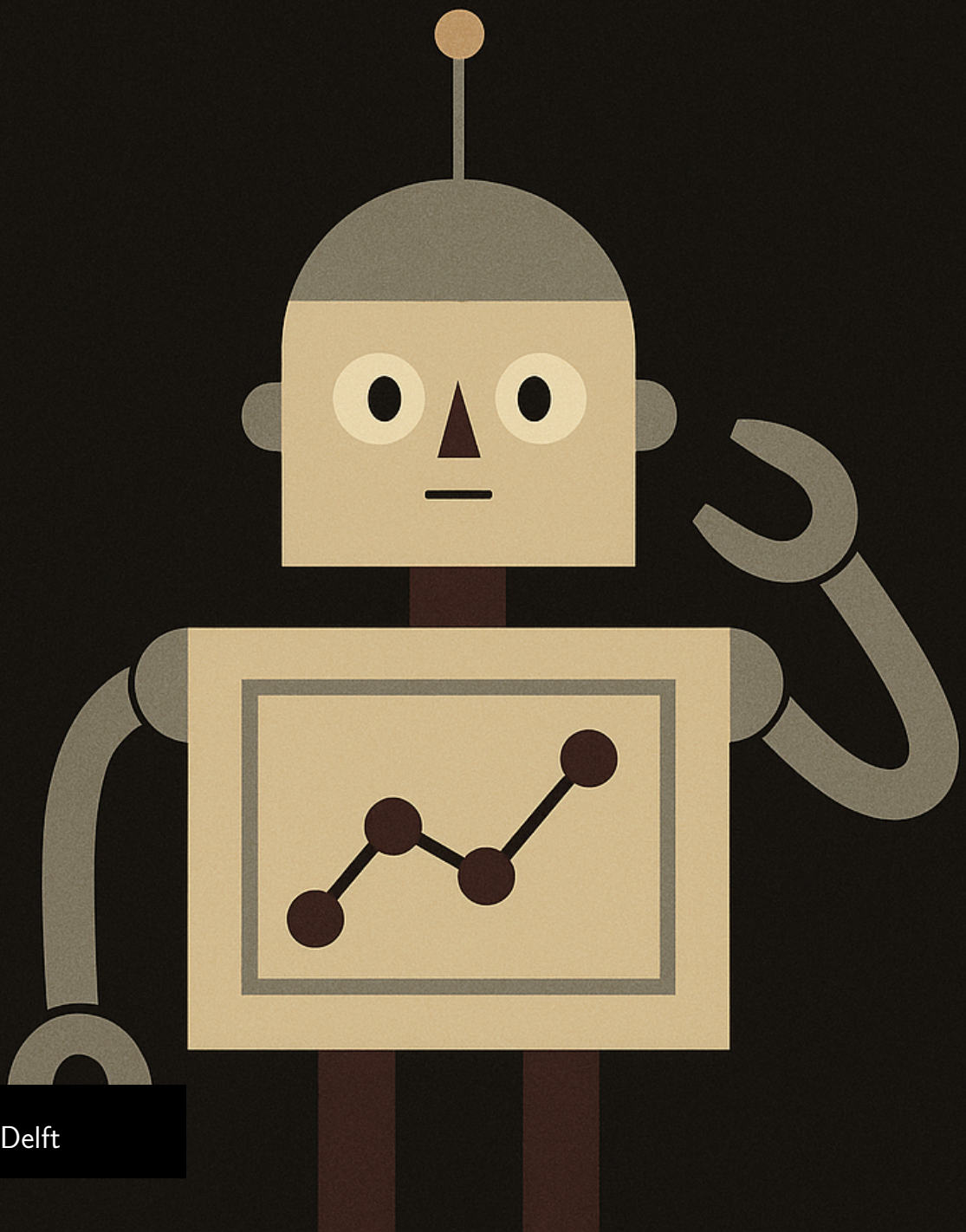
EfficientTDMPC

Improved MPC Objectives for Sample-Efficient Continuous Control

Master's Thesis

Thomas Evers



Delft University of Technology

TU Delft

Title Information

EfficientTDMPC: Improved MPC Objectives for Sample-Efficient Continuous Control

Thomas Evers (5310121)

Document type	Master's Thesis
Institution	Delft University of Technology
Faculty	Electrical Engineering, Mathematics and Computer Science
Department	Signals and Systems
Location	Delft, The Netherlands
Date	Monday 15th June, 2026

Abstract

Model-based reinforcement learning can improve sample efficiency by learning a model of the environment and using it for planning. In continuous-control tasks, this allows an agent to evaluate many candidate action sequences before acting. However, planning with a learned model also introduces a failure mode: the planner optimizes a learned return estimate rather than the true environment return, and can therefore exploit errors in the dynamics model, reward model, or value function. This thesis studies how the model predictive control objective used in TD-MPC-style agents can be made more reliable. The main contribution is EfficientTDMPC, a method that modifies the planner objective by aggregating return estimates across multiple dynamics heads and rollout depths, and by applying disagreement-based pessimism during reanalyze. These changes aim to reduce the variance and exploitability of model-based return estimates while preserving the sample efficiency benefits of latent model-based planning. EfficientTDMPC is the new state-of-the-art on HumanoidBench-Hard and the hard DeepMind Control Suite (DMC) while matching the state-of-the-art on Easy DMC. The thesis also discusses adaptive horizon selection as a future direction, arguing that planning depth should be treated as part of the uncertainty-aware planning objective. Overall, the thesis shows that the reliability of the learned planning objective is a central design problem in sample-efficient model-based reinforcement learning.

Graduation Committee

Author	Thomas Evers
Student number	5310121
Degree programme	Master of Science
Institution	Delft University of Technology
Faculty	Electrical Engineering, Mathematics and Computer Science
Department	Signals and Systems
Thesis title	EfficientTDMPC: Improved MPC Objectives for Sample-Efficient Continuous Control
Subtitle	Reducing the variance of action-value estimates in model-based reinforcement learning
Date	Monday 15$^{\text{th}}$ June, 2026

Chair	Justin Dauwels
Daily supervisor	Yaniv Oren
Supervisor	Justin Dauwels
Committee member	Wendelin Böhmer
Committee member	Cristian Meo

Preface

Its been a long road to get to this point but I wouldnt have it any other way. Sometimes I think I may one of the luckiest guys in the world.

I want to thank my parents for giving me the foundation, freedom and trust I needed to find my own way, no matter the paths I took to get here.

I want to thank Nic for being the best friend a guy could ask for, and constantly pushing me to get more out of life. Also for introducing me to tech and robotics, without you I may have become a salesmen.

I want to thank Varun for being a great friend and an inspiration. I am jealous of the future students that will get to have you as their professor :)

I want to thank Ahmad, for slowly becoming the closest thing I have to a social life in these last few months. Youre a good roomie and friend.

I want to thank Justin for giving me the first real introduction to deep learning in the project of his course, it was career changing. Also thank you for the supervision and trust in me to get things done in the end.

I want to thank Cristian, for always being there to help me out, share lessons, and support me. If you didnt choose to believe in me, this thesis would not have been possible.

I want to thank Wendelin for the conversations we have had, I gained a lot of insight, and im not sure if there is anything better than that.

Lastly, I want to thank Yaniv. Even though I now profusely sweat whenever I hear the word agenda or process, I am grateful for our discussions, your advice and your support. I hope we get to work together more in the future.

I also want to thank people who I have had the pleasure of speaking to during this project. This includes the authors of TDMPC2 and BMPC, Nicklas Hansen and Yuhang Wang, as well as some other authors in the field such as Maxime Burchi, Andrii Zadaianchuck and Claas Voelcker.

Thomas Evers
Delft, The Netherlands
Monday 15th June, 2026

Thesis Overview

This thesis studies sample-efficient model-based reinforcement learning for continuous control. It focuses on TD-MPC-style agents, where action selection depends on optimizing learned model-based return estimates. The main contribution is EfficientTDMPC, which improves the reliability of these planning objectives through ensemble dynamics, multi-depth returns, and uncertainty-aware reanalyze planning. The thesis also proposes adaptive horizon selection as a direction for future work and concludes with a broader discussion of the method's implications and limitations.

Contents

List of Figures

Nomenclature

RL Reinforcement learning.

MBRL Model-based reinforcement learning.

MPC Model predictive control.

MPPI Model predictive path integral.

DMC DeepMind Control Suite.

BMPC Bootstrapped Model Predictive Control.

TD-MPC Temporal Difference Model Predictive Control.

UTD Update-to-data ratio, the number of gradient updates performed per environment step.

Reanalyze Refreshing learning targets by rerunning planning on states sampled from the replay buffer.

AUC Area under the learning curve, used as a sample-efficiency metric.

1

Introduction

Intelligence is something humanity has long aimed to understand, and more recently, to create. Just as the industrial revolution changed the physical world, progress in artificial intelligence has the potential to change how we work, learn, and solve problems. The most successful recent attempt to build intelligent systems rests on four ideas: deep learning, self-supervised learning, transfer learning, and reinforcement learning.

1.1. Deep Learning: Learning from Data

Deep learning currently acts as the foundation of modern AI. At its core, it learns a model in the form of a deep neural network that predicts an output given an input. It does this by learning from training data. In supervised deep learning, this data consists of inputs such as images, paired with labels such as the name of the object in the corresponding image. During training, the model is changed bit by bit so that its predictions move closer to the desired outputs. After enough training, a model may predict the training data very well. However, the real goal is not merely to memorize the training data, but to make good predictions on new data that it has never seen before. As it turns out, with suitable architectures, optimization methods, and enough data, these models can generalize well to new examples [24]. Famously, AlexNet showed in 2012 that deep learning could perform extremely well on image recognition [24]. By training on millions of image-label pairs, the model recognized objects in unseen images far better than previous methods. A central lesson of this period was that more data, larger models, and more compute could lead to better generalization [23, 20]. This made data availability one of the central bottlenecks for building more capable systems.

However, labeled data is often expensive and scarce, especially for a specific task. We would much rather use unlabeled data, of which there is usually far more. This raises the question: how can a model learn from unlabeled data, and still become useful for a concrete task later?

1.2. Self-supervised and Transfer Learning: Learning From Unlabeled Data

Instead of training the model on the final task directly, we can first train it on a pretraining task that does not require hand-labeled data. The purpose of this task is to make the model learn the structure of the data itself. A common pretraining task is to use one part of a data point to predict another part. In language modeling, this can mean predicting missing or future words; in image modeling, it can mean predicting missing pixels or patches [2].

At first, such a system may not seem useful by itself; it will just generate data that is similar to what it was trained on. However, such systems have learned to understand the data very well, which can be leveraged for downstream tasks. As a thought experiment, consider a language model trained to predict the next word in a detective story. If the final word reveals who the killer was, a good next-word predictor would need to connect clues from across the story to make the right prediction. In this way, pretraining can encourage models to learn useful structure from raw data. Still, a model that only predicts missing pieces of data is not the final goal. We want to reuse this learned structure for tasks we actually care about. This is where transfer learning comes in. A model can first be pretrained using self-supervised learning, and then fine-tuned on a specific task using a smaller amount of high quality/labeled data. These systems usually work very well, because they were able to leverage large amounts of unlabeled data, and more data usually leads to better deep learning performance [23, 20].

1.3. Reinforcement Learning: Learning From Feedback

Orthogonally, reinforcement learning deals with maximizing the reward earned from interacting with an environment. It is a learning paradigm that is different from both supervised and self-supervised learning. As discussed above, supervised learning is used when we have examples that we want to imitate. This can be very effective, but it is also limited by the examples provided. A supervised learner could learn to play like a chess grandmaster by imitating them, but imitation alone cannot yield a player that plays better than the grandmaster. Reinforcement learning is not limited to imitation. If a task provides a useful feedback signal, a reinforcement learning agent can learn to take actions that maximize that feedback in the long term. This allows reinforcement learning to achieve superhuman performance in many games, such as chess, Go, and video games [39, 43, 30].

In a general form, reinforcement learning considers an environment that returns feedback in the form of a reward. The goal is to learn a policy that acts in the environment to maximize the total expected reward over an extended timeframe. This task is made difficult by two facts. Firstly, the environment is generally unknown. The agent does not know in advance what state or reward will follow from an action. It must interact with the environment, observe what happens, and learn from that experience. Crucially, the agent must also explore: it has to try actions that may be unfamiliar in order to discover whether they lead to higher reward. This contrasts with supervised learning, where the label gives a direct training signal for each example. Secondly, the action chosen does not only affect the immediate reward, but also the state of the environment, which then affects future rewards. Therefore choosing the actions that maximize the total future reward is not just a matter of maximizing the immediate reward, but also of understanding how actions taken affect the future state.

An example that well illustrates the reinforcement learning problem is the following. An AI is placed in a room and is able to move up, down, left, or right at each timestep. There is some target location in the room, and at each timestep the AI receives a reward that is higher the closer it is to the target. The issue is that the target location is behind another wall in the room, and the only way to get to the target is to first move away from it, then around the wall, and finally towards the target. In such a scenario, any system that greedily maximizes the immediate reward would walk directly towards the target, get stuck at the wall, and never learn to reach the target. A reinforcement learning agent, on the other hand, will explore and, at some point, walk around the wall and fully reach the target. It will then reinforce that behavior so that it earns higher reward consistently. The details of how this is done are explained in the background chapter, but the key insight is that reinforcement learning agents can learn to take actions that maximize the long-term sum of rewards.

1.4. Language Models as a Case Study

The ideas of deep learning, self-supervised learning, transfer learning, and reinforcement learning can be combined into a very powerful system. A good example is large language model assistants, such as ChatGPT [2, 34].

First, they are pretrained on a large corpus of text [2]. They are then fine-tuned or post-trained to act as AI assistants. This is done by providing examples of assistant-like behavior and training the model to match that behavior [34]. After this, reinforcement learning can be used to improve the model beyond the specific examples that were provided, in some cases reaching expert-level or superhuman performance on selected reasoning benchmarks [29, 21, 12]. This is done by providing feedback on the model's responses and training it to maximize that feedback. Language model assistants map to the reinforcement learning setting as follows: The state of the environment is the question that is asked, together with the rest of the conversation context. The answer the model gives is the action. The feedback it receives is the reward. Interestingly, the next state is often not modeled explicitly in this setup, meaning that the model is actually just trained to maximize the immediate reward, which is a very simplified version of the reinforcement learning problem. [1] This raises a natural question: where does the reward come from, and what should the assistant be maximizing? Two paradigms of feedback are commonly used, often in combination. Firstly, reinforcement learning with human feedback (RLHF) trains the model to give answers that score highly under human-provided feedback. This is used to make language model assistants more aligned with human preferences [7, 34]. In principle, humans could directly provide a reward for every answer. In practice, human preferences are often used to train a separate reward model. This reward model takes a question and answer as input and predicts a reward. Reinforcement learning is then used to train the language model assistant to maximize this predicted reward. Secondly, reinforcement learning with verifiable rewards (RLVR) trains the model on tasks where the answer can be checked automatically [45, 47, 12]. For example, many math and programming tasks have answers that can be verified. Other tasks, such as acting as a therapist or giving open-ended advice, are much harder to verify. When verification is possible, the reward can be more objective and scalable.

In the context of language model assistants, reinforcement learning has been the final step in the learning pipeline, and has enabled substantial performance improvements, including expert-level and in some cases superhuman results on selected benchmarks, by maximizing available reliable feedback signals. For language models, a main limitation on performance is the quality, quantity and diversity of the feedback data.

[1] A conversation with a user could be interpreted as interaction with an environment, where the user is part of the environment. This is complicated: the state would also need to contain the user's internal state, which is not directly observable, and the reward would need to be inferred from the user's response.

1.5. Toward Physical Intelligence

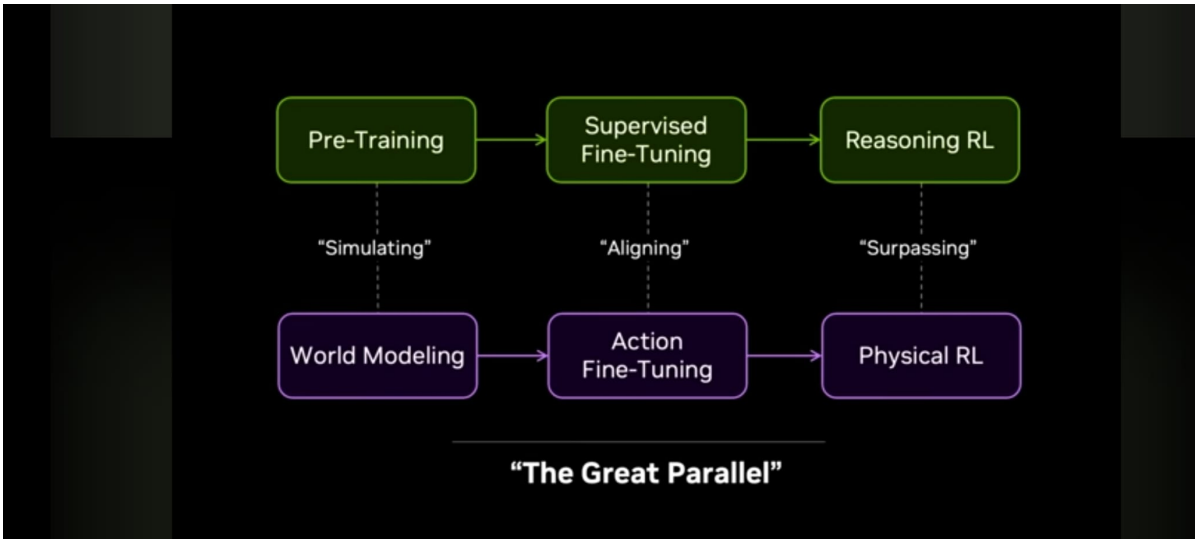


Figure 1.1: The revolution of physical intelligence may follow a similar roadmap to the revolution of language intelligence. To achieve superhuman physical intelligence we will require reinforcement learning as the final stage of the process. Image taken from a talk by Jim Fan from Nvidia on the endgame of robotics. [9]

Physical intelligence is expected to follow the same general learning pipeline that has been successful for language models. And just as in language models, reinforcement learning will be responsible for the refinement step of the process, in which superhuman physical intelligence is achieved. First, we could pretrain *world models* to understand the structure of the physical world. Instead of learning to model text, such systems would learn to model videos, sensor streams, and ideally the actions that caused them [13, 15, 16, 25]. Once a model understands how the physical world behaves and how actions affect it, the next step is to learn which actions achieve useful goals. This can first be done by imitating demonstrations, just as language models are often first adapted by imitating examples of desired behavior. The final step is to use reinforcement learning so that the system can improve beyond the demonstrations it was given. If there is a feedback signal that indicates how well a task is being completed, reinforcement learning can optimize behavior with respect to that feedback. In the long term, this could help produce robots that are highly capable at physical tasks such as cleaning, cooking, laboratory work, or industrial manipulation. The industrial revolution helped humanity automate repetitive physical labour. Computers helped humanity automate repetitive intellectual labour. The internet helped humanity communicate. Large language models have begun to automate and accelerate many language-based intellectual tasks. One of the next major steps in humanity's development is to build systems that can also learn and act effectively in the physical world, and for this, reinforcement learning will play a central role.

1.6. Motivation

This thesis aims to develop methods that reduce the amount of environment interactions required to learn to perform physical control tasks. This is also known as increasing the sample efficiency. This is an important issue to work on, since many experts agree that the physical intelligence revolution is coming soon [9]. This means many physical systems will be deployed into the world in the near future. To maximize the benefits of these systems, we will want to train them to perform their tasks at superhuman levels, which will require reinforcement learning. However, unlike in language, reinforcement learning in the physical world can be very dangerous and expensive, since it requires exploration. It is therefore

vital that we develop reinforcement learning methods that can learn to perform physical tasks with as little environment interaction as possible. Increasing the sample efficiency of our reinforcement learning algorithms on physical control has the potential to make the future of robotics cheaper, safer, and more effective.

1.7. Scope and Research Question

This thesis does not attempt to solve pretrained physical world modeling or real-world robotics; instead, it focuses on the reinforcement learning part of the physical intelligence problem. It evaluates and tests ideas on simplified environments and corresponding baselines. This is a deliberate restriction. Real-world robot learning introduces many additional challenges at once: perception, hardware reliability, safety constraints, simulation-to-reality transfer, the cost of collecting physical interaction data, and the computational costs of modeling the complexity of the real world. Studying all of these at the same time would make it difficult to isolate the core reinforcement learning methods explored in this thesis. We test our ideas on continuous-control benchmarks with fully observable, low-dimensional state descriptions. We use the DeepMind Control Suite [42] and HumanoidBench [38]. These benchmarks contain a variety of tasks in which an agent controls a simulated body to achieve a goal. Simple tasks require balancing a pole or moving a small robot, while more complex tasks require high-dimensional humanoid control, such as running or walking up stairs. These environments are much simpler than the real world, but they preserve some of the core challenges of learning to control physical systems through interaction. The choice of baseline is also part of the scope. We build on BMPC [46], a recent state-of-the-art method on the hardest tasks of the DeepMind Control Suite in terms of sample efficiency. BMPC belongs to the TD-MPC family of model-based reinforcement-learning methods [19, 18], meaning that it learns a model of the environment and uses that model during action selection and learning. This makes it a useful starting point for the thesis: if future physical-intelligence systems are likely to rely on learned world models, then improving the sample efficiency of simplified model-based baselines is a meaningful intermediate step. The broader question that this thesis aims to provide insights into is, howshould learned models be used by reinforcement learning agents to find actions that lead to high future reward? We then aim to provide insights into this question by limiting the thesis to a narrower scope, how can the model-based planning objective in TD-MPC-style agents be improved to increase their sample efficiency? This thesis develops methods and tests them to gain insights into these questions.

1.8. Approach and Contributions

This thesis approaches the research question through EfficientTDMPC, a modification of BMPC that changes how learned model predictions are used when selecting actions. The central idea is that a learned model should not be trusted as a single, fixed prediction of the future. Instead, the agent should use the model in a way that is more robust to prediction errors and uncertainty.

EfficientTDMPC makes three main changes. First, it learns multiple environment models and evaluates candidate actions across them, so that action selection does not depend too strongly on one model's prediction. Second, it evaluates imagined futures over multiple time spans, so that actions are not chosen only because they look good after one particular amount of imagined time. Third, it estimates uncertainty in the predicted consequences of actions, and adds the option to avoid actions whose predicted outcomes are too uncertain.

These changes are tested by applying them to BMPC on the DeepMind Control Suite and HumanoidBench. The experiments show that EfficientTDMPC improves sample efficiency significantly, compared to the BMPC baseline. Furthermore, to our knowledge, in the low-data regime, EfficientTDMPC is the new state of the art on DMC hard and HumanoidBench benchmarks, while matching the state of the

art on DMC easy. It is also the only method to have obtained the state of the art on all three of these benchmarks at the same time.

1.9. Thesis Roadmap

Chapter 2 introduces the technical background needed for the thesis. It first formalizes the reinforcement-learning setting and the actor-critic methods that are commonly used for continuous control. It then introduces model-based reinforcement learning, and specifically TD-MPC2 and BMPC, where BMPC is the direct baseline used for the proposed method. This background explains the method family in which the thesis question is studied. Chapter 3 presents EfficientTDMPC, the main contribution of the thesis. The chapter describes the proposed changes to model-based planning, evaluates them on DMC and HumanoidBench, and analyzes their effect on sample efficiency and computational cost. Chapter 4 discusses adaptive planning horizons as future work. This chapter extends the same central theme: learned model predictions are useful, but the agent must decide how far into the future to trust them, and how uncertainty should affect that decision. Chapter 5 concludes the thesis by discussing what the EfficientTDMPC results suggest about uncertainty-aware model-based reinforcement learning, what limitations remain, and how these lessons may contribute to future physical-intelligence systems.

2

Background

2.1. Reinforcement Learning Setup

Reinforcement learning studies agents that learn to act through interaction with an environment. In continuous-control problems, this interaction is often the expensive part of learning: a transition may require real time, hardware wear, expert supervision, or exposure to unsafe behavior. Sample efficiency is therefore not only a benchmark metric but also a practical constraint on whether a method can be used in settings such as robotics and embodied control.

We formulate the control problem as a discounted Markov decision process [41],

$$(\mathcal{S}, \mathcal{A}, P, r, \gamma).$$

The state space is denoted by $\mathcal{S} \subseteq \mathbb{R}^n$, and states are written as $s \in \mathcal{S}$. The action space is denoted by $\mathcal{A} \subseteq \mathbb{R}^m$, and actions are written as $a \in \mathcal{A}$. The transition kernel $P : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ specifies a distribution over next states, the reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ specifies the immediate reward, and $\gamma \in [0, 1)$ is the discount factor.

At time t, the agent observes s_t, selects an action a_t, receives reward

$$r_t = r(s_t, a_t),$$

and the next state is sampled as

$$s_{t+1} \sim P(\cdot \mid s_t, a_t).$$

A single environment transition is written as

$$(s_t, a_t, r_t, s_{t+1}),$$

and experience collected over time can be stored in a replay buffer,

$$\mathcal{D} = \{(s_t, a_t, r_t, s_{t+1})\}_{t=0}^{T-1}.$$

A policy maps states to distributions over actions,

$$\pi : \mathcal{S} \to \Delta(\mathcal{A}), \qquad a_t \sim \pi(\cdot \mid s_t).$$

The discounted return given some trajectory $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots)$ is the sum of discounted rewards.

$$G_t = \sum_{u=0}^{\infty} \gamma^u r(s_{t+u}, a_{t+u}), \tag{2.1}$$

The state-value function of a policy is the expected discounted return obtained by starting from state s and taking actions sampled from the policy π:

$$V^\pi(s) = \mathbb{E}_{\pi,P}\left[\sum_{u=0}^{\infty} \gamma^u r(s_u, a_u) \mid s_0 = s, a_u \sim \pi(\cdot \mid s_u), s_{u+1} \sim P(\cdot \mid s_u, a_u)\right]. \tag{2.2}$$

The expectation in Eq. (2.2) is over all future actions and states induced by the policy π and the transition kernel P. The discount factor γ is a hyperparameter that ensures the infinite discounted sum in Eq. (2.1) converges and gives more weight to near-term rewards.

The state-action value function fixes the first action and then follows the policy afterwards. Using the state-value function after the first transition, it is defined as

$$Q^\pi(s, a) = r(s, a) + \gamma\, \mathbb{E}_{s' \sim P(\cdot|s,a)}\left[V^\pi(s')\right]. \tag{2.3}$$

Only the first action is fixed to a. After the transition to s', the value $V^\pi(s')$ in Eq. (2.3) assumes that all subsequent actions are sampled from π. This is why Q^π is a value for taking action a once and then continuing according to the policy.

The optimal policy is then defined as the policy that maximizes the value function for all states,

$$\pi^* = \arg\max_\pi V^\pi(s) \quad \text{for all } s \in \mathcal{S}, \tag{2.4}$$

The optimal value functions V^* and Q^* are the value functions of the optimal policy π^*.

2.2. Simple Actor-Critic Method

In this section we describe a simple and relevant model-free reinforcement learning method. It aims to convey the fundamental ideas behind reinforcement learning, as well as some specifics that form the basis of the baselines used in this thesis. The method is a simplified version of the Soft Actor-Critic (SAC) algorithm [14], a commonly used model-free actor-critic method for continuous control. Actor-critic methods are named as such because they consist of two main components: an actor, which is the policy, and a critic, which is a value function. These components are trained by acting in the environment and learning from the collected data. The goal is to learn an optimal policy π^*, or at least a policy with high expected return. The rest of this section introduces the details of a simplified but relevant actor-critic method applied to continuous-control environments [41].

Components The main components of a model-free actor-critic method are the actor and the critic. The actor is the policy π_θ, and the critic Q_θ^π approximates the true value function Q^π for the current policy π.

Acting in the environment. Data collection begins by placing the agent in an initial environment state s_0. At each timestep, the policy samples an action

$$a_t \sim \pi_\theta(\cdot \mid s_t),$$

which is executed in the environment. The environment returns a reward and next state,

$$r_t = r(s_t, a_t), \qquad s_{t+1} \sim P(\cdot \mid s_t, a_t),$$

giving the transition (s_t, a_t, r_t, s_{t+1}). These transitions are stored into the replay buffer $\mathcal{D}$ and later sampled to update the value and policy networks.

Learning the value function. The value function can be learned by learning to match the return that is actually earned in the environment. For a learned state-action value function Q_θ^π, the objective can be written as

$$\mathcal{L}_Q(\theta) = \mathbb{E}_{(s_t, a_t, r_t, s_{t+1}) \sim \mathcal{D}} \left[\left(Q_\theta^\pi(s_t, a_t) - y_t \right)^2 \right], \tag{2.5}$$

where y_t is the value target. A common option for the value target is a one-step temporal-difference target, which uses the fact that the value of a state should be consistent with the reward just earned plus the discounted value of what comes next [1]. Using a transition (s_t, a_t, r_t, s_{t+1}) from the real environment, the 1-step TD target for $Q_\theta^\pi(s_t, a_t)$ can be written as

$$y_t^\pi = r_t + \gamma \, \mathbb{E}_{a' \sim \pi(\cdot \mid s_{t+1})} \left[Q_{\bar{\theta}}^\pi(s_{t+1}, a') \right], \tag{2.6}$$

where $\bar{\theta}$ indicates a target network, which is a slowly updated copy of the value network parameters θ that provides a stable target for learning [28]. Training Q_θ^π to match the TD target in Eq. (2.6) by minimizing Eq. (2.5) encourages the value function to be self-consistent[1]. Under certain conditions, applying this update is guaranteed to converge to the true value function Q^π for the current policy π. In practice, convergence is not guaranteed due to function approximation and finite data, but TD learning is still a powerful method for learning value functions in model-free reinforcement learning. It is worth noting that this value target for state-action values is valid for all transitions in the replay buffer, even if the action a_t was produced by an older policy. This means the use of Q values allows for *off-policy* learning, meaning the agent can learn from data collected by a different policy [28, 41]. This is not the case if V values are used.

Learning the policy. A common approach to learning a policy for continuous control in RL is to learn a stochastic policy trained to maximize the value. The policy is commonly parameterized as a tanh-squashed diagonal Gaussian [14], as shown in Eq. (2.7). A neural network outputs the mean and standard deviation of a Gaussian distribution, an unconstrained action sample is drawn from this distribution, and the final action is obtained by applying tanh so that it lies in the bounded action range.

$$a_t = \tanh(u_t), \qquad u_t \sim \mathcal{N}\left(\mu_\theta(s_t), \operatorname{diag}\left(\sigma_\theta(s_t)^2 \right) \right). \tag{2.7}$$

The policy parameters are updated to minimize a loss with two terms. The first encourages the policy to output action distributions that produce high estimated value. The second maximizes the entropy of the action distribution, encouraging the policy to stay broad enough to try new actions [14]. This gives the entropy-regularized objective

$$J(\theta) = \mathbb{E}_{s_t \sim \mathcal{D},\, a_t \sim \pi_\theta(\cdot \mid s_t)} \left[\underbrace{-Q_\theta^\pi(s_t, a_t)}_{\text{value maximization}} \; \underbrace{+ \alpha \log \pi_\theta(a_t \mid s_t)}_{\text{exploration}} \right], \tag{2.8}$$

where $\alpha \geq 0$ controls the strength of the entropy term. In Eq. (2.8), the value term guides the policy towards actions with high estimated value, while the entropy term keeps the action distribution wider

and supports exploration.

Summary: Actor-Critic

The general actor-critic agent consists of an actor, which is the policy, and a critic, which is a value function. An agent alternates between acting in the environment and storing that data in a replay buffer, and using replay-buffer data to update its components. The value network is trained to assign values to state-action pairs that are self-consistent with the trajectories in the replay buffer. The policy is then trained to output action distributions that have high estimated value under the learned critic, while often also maximizing policy entropy to encourage exploration. By alternating data collection, value learning, and policy learning, the actor-critic agent is able to learn a policy that achieves high return in the environment.

2.3. TD-MPC2

TD-MPC2 is a recent model-based actor-critic method for continuous control [18]. It keeps the central actor-critic structure from the previous subsection: it learns a policy and a state-action value function from replayed experience. The main difference is that TD-MPC2 also learns a latent dynamics model, and uses that model to change how actions are selected in the environment. Instead of sampling actions directly from the learned policy, TD-MPC2 plans over action sequences with the learned model and uses the resulting planned action distribution for acting.

This makes TD-MPC2 a hybrid between model-free actor-critic learning and model-based control. The critic is still a learned Q function, and the policy is still trained from critic feedback. However, the data entering the replay buffer is produced by a planner that uses the learned model, reward predictor, policy, and value ensemble together. The planner therefore acts as an expert policy whose actions should be stronger than actions sampled directly from the current policy network, especially when the policy is still imperfect.

Components. TD-MPC2 operates in a learned latent space. The encoder maps the observed state to a latent state,

$$z_t = h_\theta(s_t).$$

The latent representation must preserve the information from s_t that is relevant for predicting future rewards, future latent states, and useful actions. The dynamics model predicts the next latent state, assuming deterministic dynamics,

$$\hat{z}_{t+1} = f_\theta(z_t, a_t),$$

and the reward model predicts the reward earned by taking an action from a latent state,

$$\hat{r}_t = R_\theta(z_t, a_t).$$

The policy prior $\pi_\theta(\cdot \mid z_t)$ proposes actions in latent states, and the critic is an ensemble of one-step state-action value functions $Q^\pi_{\theta,i}(z_t, a_t)$ for $i \in \{1, \ldots, N_Q\}$. In TD-MPC2, $N_Q = 5$. We write Q^π_θ for the average over the value heads when the full ensemble is used. These components interact through the latent model: the encoder gives the starting latent, the dynamics and reward models score imagined action sequences, the Q ensemble bootstraps the end of those sequences, and the policy prior both guides planning and is improved from critic feedback.

Acting in the environment. In TD-MPC2, the actions taken in the environment come from an MPC-like planning procedure rather than directly from the policy network as in standard actor-critic methods. The planning procedure aims to find an action sequence distribution that, under the learned model and value function, is estimated to yield a high expected return. This expected return can be seen as the state-action value of a long-horizon action sequence. The distribution over the first action in that sequence is then defined as the planner policy π_{plan}.

The long-horizon state-action value used as planning objective is defined in Eq. (2.9):

$$\hat{Q}_H^\pi(z_t, \mathbf{a}_{t:t+H-1}) = \sum_{u=0}^{H-1} \gamma^u R_\theta(\hat{z}_{t+u}, a_{t+u}) + \gamma^H Q_\theta^\pi\left(\hat{z}_{t+H}, a_{t+H}\right), \qquad a_{t+H} \sim \pi_\theta(\cdot \mid \hat{z}_{t+H}). \tag{2.9}$$

Implementation Note:
Planning Objective Final Action from Policy

The paper describes the final action as part of the optimized sequence. The implementation optimizes the first H actions and samples the final bootstrap action from $\pi_\theta(\cdot \mid \hat{z}_{t+H})$.

where

$$\hat{z}_t = z_t, \qquad \hat{z}_{t+u+1} = f_\theta(\hat{z}_{t+u}, a_{t+u}).$$

Note that while the 1-step state-action value is estimated by the neural network Q_θ^π, the long-horizon state-action value in Eq. (2.9) is a model-based estimate that uses the learned reward and dynamics models to roll out the action sequence and then bootstraps with the learned Q at the end.

The planner then finds the action-sequence distribution that maximizes the estimated expected return as in Eq. (2.10):

$$(\boldsymbol{\mu}^*, \boldsymbol{\sigma}^*) = \arg\max_{\boldsymbol{\mu}, \boldsymbol{\sigma}} \mathbb{E}_{\mathbf{a}_{t:t+H-1} \sim \mathcal{N}(\boldsymbol{\mu}, \operatorname{diag}(\boldsymbol{\sigma}^2))} \left[\hat{Q}_H^\pi(z_t, \mathbf{a}_{t:t+H-1})\right]. \tag{2.10}$$

The details of how this MPPI-based optimization is performed can be found in the TD-MPC2 paper [18, 48]. The optimized first-step distribution defines the planner policy

$$\pi_{\text{plan}}(\cdot \mid z_t) = \mathcal{N}\left(\boldsymbol{\mu}_0^*, \operatorname{diag}\left((\boldsymbol{\sigma}_0^*)^2\right)\right).$$

In TD-MPC2, the action executed in the environment is sampled from π_{plan} rather than directly from π_θ. While this is computationally expensive, the actions produced by the planner are much stronger than actions sampled from the policy network, especially in tasks with high action dimensions [18, 46].

Learning the model. TD-MPC2 trains the model from short trajectories sampled from the replay buffer, rather than from isolated one-step transitions. We denote a sampled trajectory of horizon H by

$$\mathcal{T}_t = \{(s_{t+k}, a_{t+k}, r_{t+k}, s_{t+k+1})\}_{k=0}^{H-1}.$$

The encoder maps the observed states in this trajectory to latent states $z_{t+k} = h_\theta(s_{t+k})$ for $k = 0, \ldots, H$. These encoded latents provide the reference trajectory. TD-MPC2 then creates a predicted latent trajectory by starting from the encoded initial state and rolling out the replay action sequence with the

learned dynamics model:

$$\hat{z}_t = z_t, \qquad \hat{z}_{t+k+1} = f_\theta(\hat{z}_{t+k}, a_{t+k}), \qquad k = 0, \dots, H-1.$$

The dynamics loss matches each predicted latent to the corresponding encoded latent from the replay trajectory,

$$\mathcal{L}_{\text{dyn}}(\theta) = \mathbb{E}_{\mathcal{T}_t \sim \mathcal{D}} \left[\sum_{k=0}^{H-1} \rho^k \left\| \hat{z}_{t+k+1} - \text{sg}\left(z_{t+k+1}\right) \right\|_2^2 \right], \tag{2.11}$$

where $\text{sg}(\cdot)$ denotes a stop-gradient target [36, 11]. The factor ρ^k gives more weight to earlier prediction errors, where the model is closer to the encoded data and the target is less affected by compounding rollout error.

The reward model is trained on the same predicted latent trajectory. For each replay action a_{t+k}, the reward predictor receives the dynamics-predicted latent $\hat{z}_{t+k}$ and predicts the reward observed in the replay trajectory. This gives the reward loss in Eq. (2.12):

$$\mathcal{L}_R(\theta) = \mathbb{E}_{\mathcal{T}_t \sim \mathcal{D}} \left[\sum_{k=0}^{H-1} \rho^k \ell_R \left(R_\theta(\hat{z}_{t+k}, a_{t+k}), r_{t+k} \right) \right], \tag{2.12}$$

where ℓ_R is the reward prediction loss. Training the reward head on predicted latents is important because the planner will also query the reward head on latents produced by the dynamics model rather than on latents obtained by directly encoding real states.

Learning the value function. The value function in TD-MPC2 is still a one-step state-action value function. For each latent produced by the dynamics rollout, the critic predicts

$$Q^\pi_{\theta,i}(\hat{z}_{t+k}, a_{t+k}), \qquad i \in \{1, \dots, N_Q\}.$$

The target is constructed from the real trajectory latents rather than from the predicted latent used as the critic input. Given the encoded next latent z_{t+k+1}, TD-MPC2 samples the next action from the policy and applies the one-step Bellman target in Eq. (2.13):

$$y^\pi_{t+k} = r_{t+k} + \gamma \min_{j \in \mathcal{I}} Q^\pi_{\bar{\theta},j} \left(z_{t+k+1}, a'_{t+k+1} \right), \qquad a'_{t+k+1} \sim \pi_\theta(\cdot \mid z_{t+k+1}), \tag{2.13}$$

where $\bar{\theta}$ denotes delayed target-network parameters [28]. The subset $\mathcal{I}$ contains two critic heads sampled from the ensemble, so the target takes the minimum over two heads. Each critic head is trained to match this target on rollout latents, with the same temporal weighting used for model learning,

$$\mathcal{L}_Q(\theta) = \mathbb{E}_{\mathcal{T}_t \sim \mathcal{D}} \left[\sum_{k=0}^{H-1} \rho^k \frac{1}{N_Q} \sum_{i=1}^{N_Q} \left(Q^\pi_{\theta,i}(\hat{z}_{t+k}, a_{t+k}) - y^\pi_{t+k} \right)^2 \right]. \tag{2.14}$$

This separates the input used for critic learning from the target used for bootstrapping: the critic is evaluated on latents produced by the dynamics rollout, while the target is computed from the encoded latents of the replay trajectory. For planning and policy improvement, TD-MPC2 uses the average over the value heads instead of the pessimistic minimum used in the Bellman target.

Implementation Note: Predicted states are used to train reward and value

The paper describes reward and value prediction as using encoded latents. In the implementation, these predictions are made from latents produced by the dynamics rollout. This is a value-equivalent world-model formulation, aligned with the value-equivalence perspective and related MuZero-style world models [11, 36].

Learning the policy. The policy is trained on latent states from the replayed trajectories and imagined rollouts. It samples actions from $\pi_\theta(\cdot \mid z)$ and is updated to choose actions with high value under the learned Q ensemble. The policy objective in Eq. (2.15) uses the ensemble average,

$$J(\theta) = \mathbb{E}_{z\sim\mathcal{D},\, a\sim\pi_\theta(\cdot|z)} \left[-\frac{1}{N_Q} \sum_{i=1}^{N_Q} Q^\pi_{\theta,i}(z,a) + \alpha \log \pi_\theta(a \mid z) \right]. \tag{2.15}$$

Implementation Note: Incorrect Entropy Objective

The implementation of TD-MPC2 does not properly calculate the effect of the tanh squashing on the policy distribution, so the entropy term is not correctly computed. As a result, the policy is encouraged to be discrete/bang-bang rather than continuous. See the discussion in the TD-MPC2 repository issue for an implementation note and examples [17].

This is the same actor-critic idea as before, but applied in latent space and with an ensemble critic. The policy is not the only mechanism used for acting, because MPPI performs additional planning before an action is executed in the environment. However, a stronger policy prior improves the action samples available to the planner and provides better terminal actions for the multi-step return estimate.

Summary: TD-MPC2

As in other actor-critic methods, TD-MPC2 alternates between acting in the environment and sampling mini-batches from the replay buffer to update its components. The main difference is that TD-MPC2 learns a latent model of the environment, which it then uses to plan out actions that it executes in the environment, instead of the actions from the policy network. Some additional effects of learning a model are improved latent representations due to auxiliary losses, as well as a value head that is trained to be value-equivalent.

2.4. BMPC

BMPC builds on TD-MPC2, but changes the two learned components that most directly interact with the planner [46]. The policy is trained by distilling the planner into the policy network, and the state-action value functions are replaced by state-value functions trained with imagined TD targets.

Learning the value function. BMPC replaces TD-MPC2's state-action value functions $Q^\pi_{\theta,i}(z_t, a_t)$ with state-value functions $V^\pi_{\theta,i}(z_t)$. This changes how value targets can be formed. A state-action value

target can be learned from off-policy replay transitions because the sampled action a_t is part of the quantity being evaluated. A state-value target, however, should describe the value of following the current policy from a state. The next state in an arbitrary replay transition was produced by the behavior policy that collected the data, not necessarily by the current policy. BMPC resolves this by using the learned model to create a synthetic on-policy transition from a replay state.

Given a replay state s_t, BMPC encodes it as $z_t = h_\theta(s_t)$, samples an action from the current policy, and constructs the model-based TD target in Eq. (2.16):

$$y_t^\pi = R_\theta(z_t, a_t^\pi) + \gamma V_{\bar{\theta}}^\pi(\hat{z}_{t+1}), \qquad a_t^\pi \sim \pi_\theta(\cdot \mid z_t), \tag{2.16}$$

where the next latent is imagined by the learned dynamics model,

$$\hat{z}_{t+1} = f_\theta(z_t, a_t^\pi).$$

The important difference from TD-MPC2 is therefore not a new value-loss form, but a new value target: BMPC trains state values against TD targets generated from replay states, current-policy actions, and model-predicted next latents, rather than from the true replay action and next state.

Learning the policy. BMPC trains the policy by distilling planner outputs stored in the replay buffer. When the planner is run on a latent state, the first-action marginal of the optimized action-sequence distribution defines an expert planner policy $\pi_{\text{plan},t}$. BMPC stores this planner distribution together with the transition and trains the policy to match the stored expert target. The policy objective is therefore a planner-distillation loss rather than a value-maximization objective. The distillation pushes $\log \pi_\theta$ towards $\log \pi_{\text{plan}}$, while the entropy term $-\alpha \log \pi_\theta$ encourages the policy to stay broad, as shown in Eq. (2.17):

$$\mathcal{L}_\pi(\theta) = \mathbb{E}_{\substack{(z,\pi_{\text{plan}})\sim\mathcal{D},\\ a\sim\pi_\theta(\cdot|z)}} \left[\alpha \log \pi_\theta(a \mid z) - \log \pi_{\text{plan}}(a)\right]. \tag{2.17}$$

In this objective, the planner is the expert and the policy network is the student. This differs from TD-MPC2, where the policy is improved by sampling actions and maximizing their predicted state-action value under the learned critic.

Implementation Note: BMPC uses the backward KL

BMPC describes the distillation loss as a forward KL from the planner to the policy. The implementation uses the reverse direction, from the policy to the planner, which makes the update mode-seeking.

BMPC also changes the policy parameterization to match the clipped Gaussian form of the planner target. Instead of using the usual tanh-squashed Gaussian policy, BMPC bounds the mean with tanh, adds Gaussian noise in action space, and clips the resulting action to the valid action range,

$$\epsilon_t \sim \mathcal{N}(0, I), \qquad u_t = \tanh(\mu_\theta(z_t)) + \sigma_\theta(z_t) \odot \epsilon_t, \qquad a_t = \text{clip}(u_t, -1, 1).$$

The clipped parameterization makes the policy distribution closer to the distribution produced by the planner, whose sampled actions are also clipped to the environment action bounds.

Summary: BMPC

BMPC is similar to TD-MPC2, except for changes in the value and policy learning. Firstly, in BMPC the policy is trained by being distilled from the planner. During acting, BMPC stores not only the transition, but also the planner policy for that transition in the replay buffer. Those planner action distributions are then distilled into the policy to train it. At some interval, some of the stale planner policies in the replay buffer are refreshed by running reanalyze planning [36]. This ensures that the policy sees recent targets. Secondly, BMPC does not use parameterized Q functions, but instead V value functions. It trains the value function using on-policy synthetic trajectories that are generated by the learned model.

3

EfficientTDMPC: Improved MPC Objectives for Sample-Efficient Continuous Control

Summary: EfficientTDMPC

This chapter presents the EfficientTDMPC paper. EfficientTDMPC introduces several modifications that aim to make the planning objective used in TD-MPC-style agents less exploitable. First, it replaces the single dynamics model with an ensemble of dynamics heads, so candidate action sequences need to yield good predictions across all dynamics and value heads. Second, it averages return estimates across multiple rollout depths, rather than just the maximum depth, further reducing the variance of the return estimates. Third, it introduces an uncertainty-aware planning objective that applies pessimism during reanalyze planning, so the planner avoids producing policy targets from action sequences whose return estimates are estimated to be uncertain. Together, these changes aim to reduce the variance and exploitability of the return estimate that the planner optimizes. Additionally, EfficientTDMPC introduces several auxiliary improvements, including increasing the UTD ratio, reducing compute during reanalyze planning, and inserting transitions into the replay buffer at every environment step rather than only at episode end. While the UTD increase and the uncertainty-aware planning showed significant improvements, the dynamics ensemble and mixed-depth return estimates did not, on their own, yield statistically significant gains. When applied jointly, however, EfficientTDMPC outperforms the baseline; uncertainty-aware planning also depends on the dynamics ensemble, and experiments indicate that the UTD scaling was partially enabled by the other modifications. Not only does EfficientTDMPC significantly outperform the baseline, but it also achieves state-of-the-art performance on the HumanoidBench-Hard and DMC-hard domains while matching the state-of-the-art on the DMC-easy domain. Among the compared methods, it is the only one to achieve state-of-the-art performance on all three domains.

3.1. Introduction

Model-based reinforcement learning (MBRL) methods that learn a world model and use it for planning have achieved strong sample efficiency in continuous control [49, 22, 8, 19, 18], with applications spanning robotics, autonomous vehicles, and spatial understanding and reasoning [15, 16, 27]. The TD-MPC family [19] has been very successful in this domain by introducing a latent dynamics model with an MPC planner to find action sequences that maximize a bootstrapped return estimate.

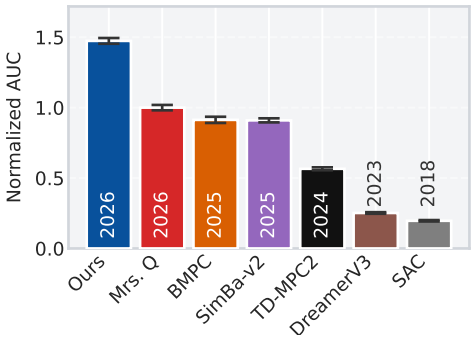


Figure 3.1: Mean normalized sample efficiency across DMC Easy, Hard and HumanoidBench-Hard. Bars show mean area under the normalized aggregated learning curves (AUC) of each benchmark; error bars show 95% CIs over 35 tasks, 3 seeds each.

Several recent works in the family, like TD-M(PC)^2 [26], BOOM [50] and BMPC [46], then distill this planner into the policy network, relying even more strongly on planning.

The planner's purpose is to find action sequences that maximize the estimated return. This estimate is created by rolling out the learned dynamics and reward models and then bootstrapping with a value network. However, when the learned model is inaccurate, the planner can exploit these inaccuracies to find action sequences that look good under the learned model but are not actually good in the environment [8, 22].

EfficientTDMPC replaces the single dynamics model with an ensemble, which previous work has shown can reduce the error in model rollouts [8, 22, 49]. We also average the return estimates across different rollout depths, which has been successful in methods using MCTS [3, 39, 36], and is known to reduce the variance of model-based return estimates [31, 3]. Lastly, EfficientTDMPC introduces the option to apply a pessimistic penalty to the planner objective, which penalizes action sequences for which the return estimate is uncertain. EfficientTDMPC obtains the strongest aggregate performance among compared methods on HumanoidBench-Hard and DMC hard, while matching the strongest baseline on DMC easy.

This paper contributes

- **Improved planner objectives.** Three mechanisms to reduce the variance in estimating the return of action sequences: adding a dynamics ensemble, a mixed-depth value estimate, and a pessimistic penalty in the planner objective.
- **Pipeline improvements.** These improvements include: per-step replay insertion, reduced compute, and more successful UTD scaling.
- **Performance.** Evidence that EfficientTDMPC matches or outperforms SOTA baselines on HumanoidBench-Hard, DMC hard and DMC easy, while being the only SOTA-performing method across all three benchmarks.

3.2. Background

Problem setting and notation: We consider continuous-control problems modeled as discounted Markov decision processes $(\mathcal{S}, \mathcal{A}, P, r, \gamma)$. Here, $\mathcal{S} \subseteq \mathbb{R}^n$ and $\mathcal{A} \subseteq \mathbb{R}^m$ denote the state and action spaces, where $s \in \mathcal{S}$ are states and $a \in \mathcal{A}$ are actions, $P : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is the state-transition kernel, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, and $\gamma \in [0, 1)$ is the discount factor.

A policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$ maps states to action distributions. We define the value of a policy at state s as the expected discounted return when starting from s and following π thereafter, $V^\pi(s) = \mathbb{E}_{\pi,P}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid s_0 = s, a_t \sim \pi(\cdot \mid s_t), s_{t+1} \sim P(\cdot \mid s_t, a_t)\right]$.

TD-MPC family: Our method builds on the TD-MPC family of works [19, 18] which use a strong planner to find an action sequence that maximizes the estimated return. The estimated return of an action sequence is then obtained by rolling out a latent world model and bootstrapping with a value network. The planner is used both for acting in the environment and for distilling its behavior into a policy network. Several followup works like TD-M(PC)^2, BOOM, and BMPC then distill this planner into the policy, relying even more strongly on the planner. We build on BMPC [46] specifically, since it is a recent contribution that has shown strong empirical performance.

Latent world model: Table 3.1 lists the model components of BMPC.

Table 3.1: Model components of BMPC

Component	Notation	Role
Encoder	h_θ	Maps observation s_t to latent $z_t = h_\theta(s_t)$
Dynamics model	f_θ	Predicts next latent: $z_{t+1} = f_\theta(z_t, a_t)$
Reward model	R_θ	Predicts scalar reward from a latent-action pair
Policy (prior)	$\pi_\theta(\cdot \mid z)$	Stochastic policy used to guide planning
Value network	$V_{\theta,i}(z)$	Ensemble of state-value networks $i \in \{1, .., N_v\}$

Given an observed state s_t, the encoder produces the latent representation $z_t = h_\theta(s_t)$. The dynamics head is then trained to predict the next latent $\hat{z}_{t+1} = f_\theta(z_t, a_t)$ given an action a_t. Similarly, the reward model predicts the reward earned at each step $\hat{r}_t = R_\theta(z_t, a_t)$, and each member of the value ensemble predicts the value at a state $V_{\theta,i}(z_t)$. We use $X_{\theta,i}$ to denote the i-th ensemble member, and use the notation X_θ as average over the ensemble.

Planner objective: In the TD-MPC family, a planner optimizes an action sequence of length H to maximize the expected return. The exact objective used is the state-action value of the action sequence, which is defined as the expected return of executing the action sequence $\mathbf{a}_{t:t+H-1}$ starting from s_t, and then following policy π thereafter.

$$Q_H^\pi(s_t, \mathbf{a}_{t:t+H-1}) = \mathbb{E}_P\left[\sum_{u=0}^{H-1} \gamma^u r(s_{t+u}, a_{t+u}) + \gamma^H V^\pi(s_{t+H}) \mid s_t, \mathbf{a}_{t:t+H-1}, s_{t+1} \sim P(\cdot \mid s_t, a_t)\right].$$

Since direct access to this quantity is not available, the TD-MPC family estimates the state-action value by rolling out its learned dynamics and reward models, and bootstrapping with the value network at the end of the rollout. Specifically, given a latent state z_t and an action sequence $\mathbf{a}_{t:t+H-1}$, the state-action value estimate is defined as

$$\hat{Q}_H^\pi(z_t, \mathbf{a}_{t:t+H-1}) \;=\; \sum_{u=0}^{H-1} \gamma^u R_\theta(\hat{z}_{t+u}, a_{t+u}) \;+\; \gamma^H V_\theta^\pi(\hat{z}_{t+H}), \qquad \hat{z}_{t+u+1} = f_\theta(\hat{z}_{t+u}, a_{t+u}).$$

where $\hat{z}_t = z_t$. This state-action value estimate, which we also call the return estimate, is then used as the objective for the planner.

Specifically, the MPPI planner in the TD-MPC family tries to find the action distribution that maximizes the expected estimated state-action value of the action sequence sampled from that distribution,

as follows:

$$(\boldsymbol{\mu}^*, \boldsymbol{\sigma}^*) = \arg\max_{\boldsymbol{\mu},\boldsymbol{\sigma}} \mathbb{E}_{\mathbf{a}_{t:t+H-1} \sim \mathcal{N}(\boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}^2))} \left[\hat{Q}_H^{\pi}(z_t, \mathbf{a}_{t:t+H-1}) \right]$$

where the details about the MPPI optimization can be found in the TD-MPC2 paper [18]. We then define the expert policy π_{plan} as the Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}_0^*, \mathrm{diag}((\boldsymbol{\sigma}_0^*)^2))$, where $\boldsymbol{\mu}_0^*$ and $\boldsymbol{\sigma}_0^*$ are the first timestep from $\boldsymbol{\mu}^*$ and $\boldsymbol{\sigma}^*$. All members of the TD-MPC family sample from this expert policy π_{plan} to act in the environment.

Aside from planner objectives, the 1-step imagined value targets can also be seen as state-action value estimates, but of the policy actions. Specifically, the value target y_t is calculated as $\bar{Q}_1^{\pi}(z_t, a_t)$ with $a_t \sim \pi_\theta(\cdot \mid z_t)$, where $\bar{\cdot}$ denotes that an exponential moving average of the value network was used, which is common practice. The value heads are then trained to minimize the error with respect to this value target.

Planner distillation into the policy, and reanalyze: In BMPC the expert action distribution output by the planner is also distilled into the policy network. Specifically, when an environment transition is collected the corresponding expert action distribution is stored as a policy target, and the policy is trained to minimize $D_{\text{KL}}(\pi_{\text{plan}} \,\|\, \pi_\theta(\cdot \mid z_t))$. This makes the policy act similarly to what was executed in the past. BMPC then implements reanalyze [36] which refreshes the policy targets in the replay buffer by rerunning the planner on the replay states. These reanalyze targets steer the policy towards fresher expert actions, but crucially these actions are never executed in the real environment. One issue that reanalyze causes is that the fresh policy targets have never been executed in the environment. As a result the model was not trained on them. This is a problem, since the model rolls out the policy actions to create the value targets, which may not be accurate for these actions.

3.3. Related work

TD-MPC family and planner distillation

As discussed in Section 3.2, in the TD-MPC family, the actions from the learned policy are never executed in the environment; therefore errors exploited by the policy may never be corrected through interaction with the environment. BOOM [50] and TD-M(PC)^2 [26] address this issue by regularizing the policy towards the action taken in the environment, which leads to large improvements on tasks with high-dimensional action spaces. A downside, however, is that the policy is regularized towards stale actions, which is suboptimal. BMPC [46] then suggests to train the policy fully with distillation, using reanalyze [36] to refresh the stale policy targets.

Dynamics ensembles in model-based reinforcement learning

Ensembles of learned dynamics models are widely used in MBRL. PETS [8], MBPO [22], MOPO [49] and STEVE [4] use bootstrapped dynamics ensembles for uncertainty-aware model rollouts. The ensembles reduce variance in the rollouts, but also provide an estimate of the epistemic uncertainty of the model rollout.

Pessimism in reinforcement learning

Pessimism in reinforcement learning is a strategy to avoid epistemically uncertain regions [35]. Pessimism is often implemented as a penalty to the value estimates, either by taking a minimum over an ensemble or subset of the ensemble [6, 14, 32, 18, 10] or by subtracting a disagreement based penalty [49]. This penalty can be applied to value targets [14, 6, 10], to the policy objective, or to the planner objective [8, 22, 49]. MRS.Q [5] recently showed that pessimism is critical for achieving strong performance on HumanoidBench-Hard [38], but also notes that it can hurt performance on DMC [42].

Multi-horizon value estimation

Combining value estimates across multiple rollout depths is a common strategy in (model-based) RL to reduce the variance of a return estimate. DreamerV3 [16] rolls out policy actions in its learned model and uses TD(λ) [40] to combine the return estimates from different rollout depths, yielding lower variance value estimates. MCTS-based methods like MuZero [36] and TSMCTS [31] use a uniform average over multiple rollout depths to reduce the variance of their value estimates. To our knowledge, prior work has not used a mixed-depth return estimate as the objective optimized by an MPPI-style MPC planner in the TD-MPC family.

3.4. EfficientTDMPC

We introduce EfficientTDMPC, a TD-MPC-style method that improves the reliability of model-based return estimates used for planning. The TD-MPC family relies heavily on its planner to find actions that maximize the estimated return. Reducing the error in the return estimate can be expected to yield better results from planning. The return estimate is obtained by rolling out the learned dynamics and reward models, and bootstrapping with the value network at the end of the rollout. As such, the return estimate is prone to error from all components, which compounds with rollout depth. EfficientTDMPC makes the planner objective less sensitive to errors in any single learned model rollout. First, it introduces an ensemble of dynamics models and averages over each of their predictions of the return, which previous work has shown can reduce error [8, 22, 49] of model-based return estimates. Secondly, it averages the return estimates from rolling out and bootstrapping at different depths, which has been successful in other works that employ MCTS [3, 39, 36], and is known to reduce estimator variance under weak assumptions [31, 3]. Third, it introduces the option to apply pessimism to the planner objective during reanalyze, which prevents distilling action distributions into the policy for which the value estimates are very uncertain. We then introduce two practical training improvements that improve replay buffer freshness and reduce wall-clock training time. Lastly, we find that our contributions enable us to successfully scale the UTD ratio, further improving sample efficiency. We build our contributions on BMPC [46], a recent strong contribution to the TD-MPC family.

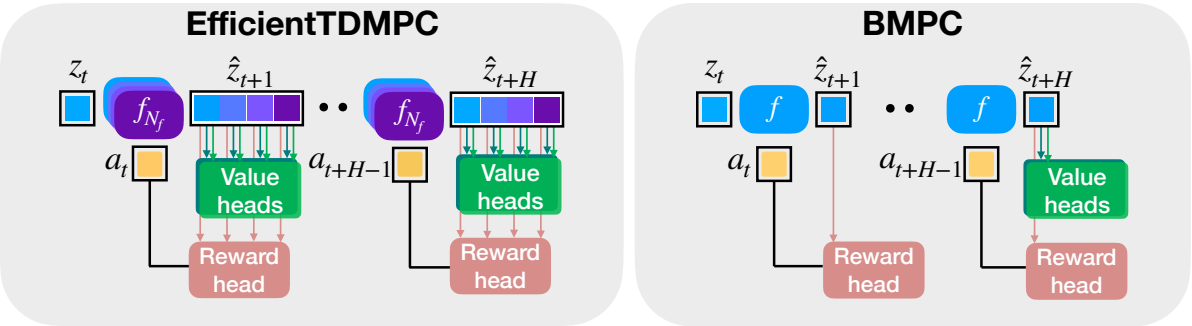


Figure 3.2: Process of return estimation of EfficientTDMPC vs BMPC (Right) BMPC rolls out a single dynamics head, predicts the reward earned at each depth and bootstraps with its value ensemble. (Left) EfficientTDMPC creates multiple rollouts using an ensemble of dynamics models. It then predicts the reward and bootstrapped value at each depth. The estimate is then averaged over the different rollouts and the return estimates from different rollout depths.

3.4.1. Ensembles for state-action value estimation

EfficientTDMPC introduces an ensemble of dynamics models to the TD-MPC family, denoted as $f_{\theta,i}$ for $i \in \{1, .., N_f\}$,where N_f is the number of dynamics heads. Prior work has shown that dynamics ensembles can reduce the error in model based return estimates [8], but this has not been applied to the TD-MPC family. We propose to introduce a method similar to the TS_∞ trajectory sampling method

introduced by PETS, in which each dynamics head is rolled out individually, and the resulting return estimates are combined. The main difference from PETS is that EfficientTDMPC bootstraps each rollout with learned value networks.

We first define a state-action value estimate that uses a single dynamics head with index $i \in \{1, .., N_f\}$ and a single value head with index $j \in \{1, .., N_v\}$ as,

$$\hat{Q}^{\pi}_{H}(z_t, \mathbf{a}_{t:t+H-1}; i, j) \;=\; \sum_{u=0}^{H-1} \gamma^u R_\theta(\hat{z}_{t+u}, a_{t+u}) \;+\; \gamma^H V_{\theta,j}(\hat{z}_{t+H}),\; \hat{z}_{t+u+1} = f_{\theta,i}(\hat{z}_{t+u}, a_{t+u}) \tag{3.1}$$

where $\hat{z}_t = z_t$. This is the state-action value estimate obtained by rolling out the dynamics head with index i and bootstrapping with the value head with index j. We can then obtain $N_f \times N_v$ different estimates of the state-action value by varying i and j. We then define our improved state-action value estimate as the average over each combination of dynamics and value heads,

$$\hat{Q}^{\pi}_{H}(z_t, \mathbf{a}_{t:t+H-1}) = \frac{1}{N_f N_v} \sum_{i=1}^{N_f} \sum_{j=1}^{N_v} \hat{Q}^{\pi}_{H}(z_t, \mathbf{a}_{t:t+H-1}; i, j) \tag{3.2}$$

We also use the ensemble disagreement to estimate the variance of the state-action value estimate. The variance estimate used is the standard estimator of the variance of the mean, given $N_f \times N_v$ independent samples:

$$\hat{\sigma}^2_{\hat{Q}^{\pi}_{H}}(z_t, \mathbf{a}_{t:t+H-1}) = \frac{1}{N_f N_v (N_f N_v - 1)} \sum_{i=1}^{N_f} \sum_{j=1}^{N_v} \left(\hat{Q}^{\pi}_{H}(z_t, \mathbf{a}_{t:t+H-1}; i, j) - \hat{Q}^{\pi}_{H}(z_t, \mathbf{a}_{t:t+H-1}) \right)^2 \tag{3.3}$$

This variance estimate, written as $\hat{\sigma}^2_{\hat{Q}^{\pi}_{H}}$, is then used in Section 3.4.3 as a penalty to the planner objective during reanalyze. We acknowledge that this estimate underestimates the true variance because it assumes that the individual estimates are independent, which is not the case since the same value or dynamics head is used for several of them. We use this variance estimate purely as a penalty to the planner objective during reanalyze, and find that it works well in practice.

3.4.2. Aggregate multi-horizon planning objective

To further reduce estimator variance, we introduce a *mixed-horizon* planner objective, obtained by averaging the state-action value estimates across different rollout horizons. As discussed in Section 3.3, this form of value aggregation has been shown to reduce variance under mild assumptions and has been successful in other model-based RL planning methods like MCTS [3, 39, 36]. We introduce this idea to MPC-based planners within model-based RL, which, to our knowledge, has not been done before.

$$\hat{Q}_{\text{aggregate},H}(z_t, \mathbf{a}_{t:t+H-1}) = \frac{1}{H} \sum_{h=1}^{H} \hat{Q}_h(z_t, \mathbf{a}_{t:t+h-1}) \tag{3.4}$$

where each $\hat{Q}_h$ only evaluates the value of the first h actions in the action sequence, and bootstraps with the value network at depth h. If we assume the estimates $\hat{Q}_h$ are not fully correlated, or that the variance of the estimates increases with depth, then the variance of our proposed objective will be lower than the single horizon objective at that rollout depth [31]. We use the horizon-aggregated objective during planning at acting time, together with an increase in the rollout horizon.

3.4.3. Pessimistic reanalyze

We propose to add the option of pessimism to the planner objective during reanalyze, to avoid distilling actions into the policy on which the model and value networks are not well trained. As discussed in Section 3.3, a known issue in the TD-MPC family is that of a mismatch between the actions taken in the environment and the actions output by the policy network [50]. Since the model and value networks are not trained on the policy actions, they are more likely to produce inaccurate predictions for them. Prior work solved this by regularizing the policy towards the action that was taken in the environment, but in later parts of the training that action becomes stale, which can be expected to slow down learning. BMPC solves this by reanalyzing the stale targets, but by doing so it reintroduces the issue where the model may not be trained on the actions output by the policy.

EfficientTDMPC introduces a pessimistic mechanism to reanalyze, to make policy targets that are fresh, but are also regularized towards actions that the model has actually been trained on. Specifically, we define the pessimistic reanalyze objective as follows:

$$J_{\text{reanalyze},H}(\mathbf{a}_{t:t+H-1}) = \hat{Q}^{\pi}_{H}(z_t, \mathbf{a}_{t:t+H-1}) - \beta\hat{\sigma}_{\hat{Q}^{\pi}_{H}}(z_t, \mathbf{a}_{t:t+H-1}),$$

where $\hat{\sigma}_{\hat{Q}^{\pi}_{H}}(z_t, \mathbf{a}_{t:t+H-1})$ is the square root of the estimated variance defined in Section 3.4.1. Here β trades off maximizing expected return and minimizing uncertainty. A higher β leads to a planner that finds actions with more certain return estimates, which are more likely to be accurate. Since the reanalyze targets are distilled into the policy, the policy also outputs actions that lead to more certain return estimates, which in turn leads to more reliable value targets. We note that the option to add pessimism is fundamentally a tradeoff between variance reduction and maximization, to which different tasks respond differently. As mentioned in Section 3.3, prior work has found that HumanoidBench-Hard generally benefits from pessimism, while DMC tasks do not.

3.4.4. Practical modifications

Data freshness. EfficientTDMPC inserts each transition immediately, as opposed to previous work in the TD-MPC family which adds newly collected transitions after a full episode. This shortens the delay before new experience can affect model, value, and policy learning.

Cheaper reanalyze. Reanalyze is responsible for the majority of the wall-clock training time, so we reduce its planning budget by reducing the particles, elites and policy seeds by a factor of eight. We find in Section 3.5.2 that, for reanalyze, fewer planning particles are sufficient in practice. Additionally, we do not apply the longer horizon and horizon aggregation to reanalyze to save compute.

Higher UTD scaling. We find that EfficientTDMPC benefits more from increasing the UTD than BMPC does. We define scaling the UTD as proportionally increasing the number of gradient steps taken per environment step, as well as increasing the reanalyze ratio.

3.5. Experiments

We implement our EfficientTDMPC agent with an ensemble of four dynamics heads and two value heads. During planning at acting time we increase the rollout horizon from three to six and optimize the aggregate horizon objective. To save compute we do not apply the aggregate horizon objective to planning during reanalyze. For HumanoidBench we choose a reanalyze pessimism coefficient of $\beta = 10$ based on the sweep in Section 3.5.2, while for DMC we set $\beta = 0$, since our experiments in Figure 3.5 and prior work [5, 32] show that pessimism can hurt performance in this domain. We describe the baselines used for comparison in Appendix A.1.3.

3.5.1. Results.

Figure 3.3 gives the normalized aggregate learning curves of EfficientTDMPC compared to several strong baselines. In normalized aggregate learning curves, EfficientTDMPC reaches higher sample efficiency than the strongest compared baseline on HumanoidBench-Hard and Hard DMC, and remains competitive with the strongest baseline on Easy DMC.

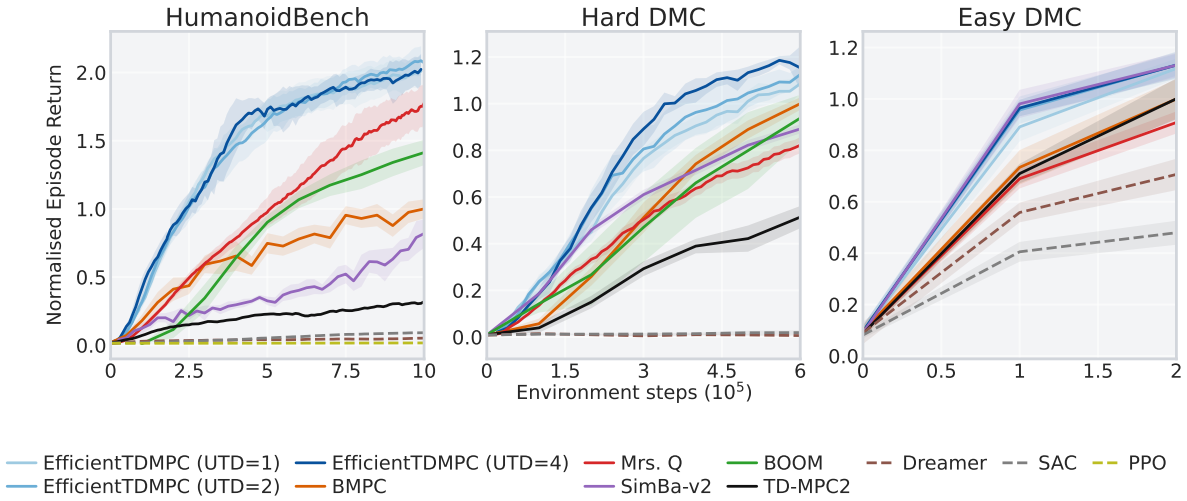


Figure 3.3: Learning Curves per benchmark. Normalized episode return aggregated over the tasks of each benchmark. (left) *HumanoidBench-Hard (7 tasks)*, (middle) *Hard DMC (7 tasks)*, (right) *Easy DMC (21 tasks)*. Shaded regions show 95% confidence intervals over at least three seeds per task. Appendices A.2.1, A.2.2, and A.2.3 provide the complete per-task learning-curve grids. Appendix A.1.2 gives the full protocol including a description of the HumanoidBench subset used

3.5.2. Component contributions

Dynamics ensemble, Buffer update, and Aggregate horizon objective. We isolate three components of EfficientTDMPC and evaluate their individual effects by adding them to the baseline (BMPC). Results are averaged over six seeds and four representative tasks that span the different action dimensions present in DMC: quadruped-walk, reacher-hard, dog-stand, and humanoid-walk. Figure 3.4 shows the task-aggregated contribution of these components to final performance. Interestingly, although the trends are generally positive, none of the isolated components clearly separate from BMPC under our confidence intervals and number of seeds used. However, we see in Figure 3.3 that when combined, EfficientTDMPC achieves a strong performance, which supports the hypothesis that the contributions may complement each other. Appendix A.2.4 shows the individual learning curves for each task.

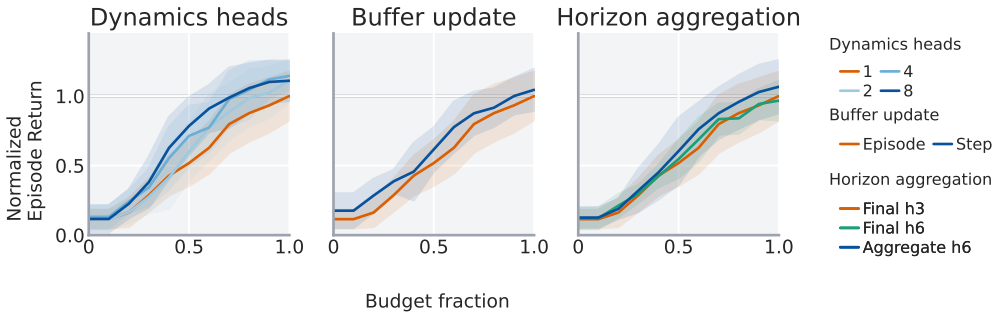


Figure 3.4: Component ablations. Left: the effect of the dynamics-ensemble size. Center: the effect of per step replay-buffer insertion. Right: The effect of the horizon aggregation. Shaded regions show 95% confidence intervals for mean normalized return across four ablation tasks with six seeds each.

Reanalyze pessimism. We ablate the effect of pessimism during reanalyze by sweeping the pessimism coefficient β over $\{0, 1, 3, 10, 30\}$ on three tasks that represent the effect of pessimism on each benchmark: h1hand-walk for HumanoidBench, dog-run for hard DMC and reacher-hard for easy DMC. Figure 3.5 shows that moderate pessimism can significantly improve performance on h1hand-walk, while having a negative or insignificant effect on the DMC tasks. This is in line with prior work [32, 5], which states that pessimism reduces overestimation bias, which can hurt performance in DMC. We therefore choose to add pessimism to tasks in HumanoidBench but not DMC. Being able to set the pessimism level for different tasks is both a feature and a limitation of our work.

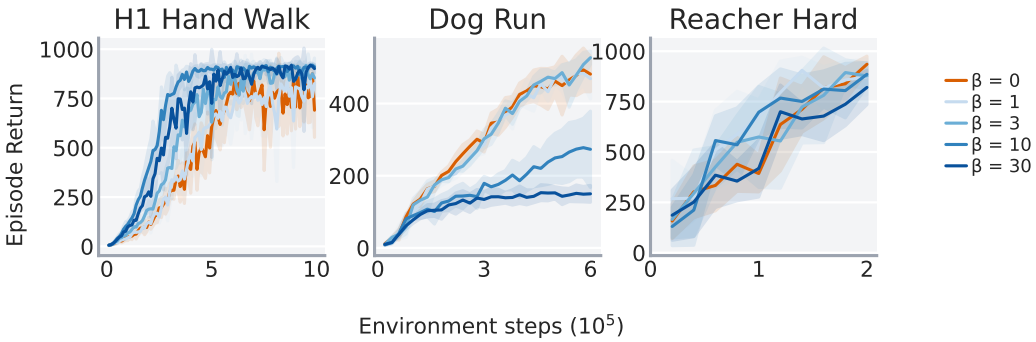
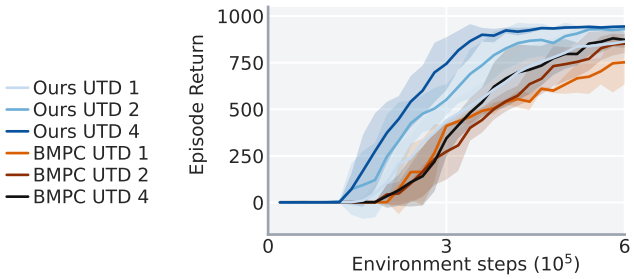


Figure 3.5: Reanalyze pessimism ablation. Evaluation reward for pessimism coefficients $\beta \in \{0, 1, 3, 10, 30\}$ on h1hand-walk, dog-run, and reacher-hard, which are chosen to show a representative effect. Moderate pessimism improves h1hand-walk, has a mixed effect on reacher-hard, and degrades dog-run at larger coefficients.

Combined Contributions and UTD scaling. As mentioned previously, the contributions of EfficientTDMPC may complement each other, yielding significant improvements for EfficientTDMPC with UTD-1 with respect to BMPC. We find that our method also benefits from increasing the UTD ratio, an example of which can be seen in Figure 3.6b. While the effect is not strong, EfficientTDMPC seems to scale better with higher UTD than BMPC does. Appendix A.2 shows the effect of UTD scaling on each task.

Method	Minutes	Rel. BMPC
BMPC	45 ± 3	1×
Ours UTD-1	47 ± 2	1.05×
Ours UTD-2	59 ± 4	1.31×
Ours UTD-4	109 ± 5	2.42×

(a)

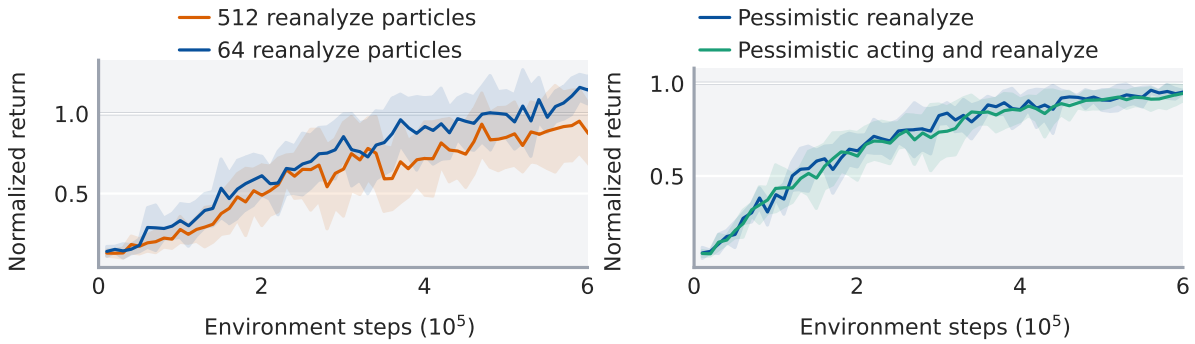


(b)

Figure 3.6: UTD scaling and runtime. (a) Minutes to train for 200k environment steps on a single NVIDIA A100 GPU for BMPC, and for EfficientTDMPC at different UTD ratios; full comparison detail is in Appendix A.1.4. (b) UTD scaling on humanoid-walk. EfficientTDMPC benefits strongly from higher UTD, while BMPC improves more gradually under the same scaling; shaded regions show 95% confidence intervals using 5 seeds per method.

UTD and wall-clock training time. We also compare the wall clock runtime of EfficientTDMPC under different UTD ratios against BMPC in Figure 3.6a. We find EfficientTDMPC with UTD 4 does have a significantly higher wall-clock training time compared to BMPC. This is a limitation of our proposed method. However we find that EfficientTDMPC with UTD-1 still outperforms BMPC while

having similar training times. This is in part due to the reduced number of MPPI particles used during reanalyze, which save significant compute, while not hurting performance, as shown in Figure 3.7a. Separately, Figure 3.7b shows that applying pessimism during acting in addition to reanalyze does not improve performance.



(a) (b)

Figure 3.7: Planner ablations. (a) Cheap reanalyze: normalized aggregate planner performance showing the effect of cheaper reanalyze, comparing 512 reanalyze particles against 64 reanalyze particles across four HumanoidBench tasks, with 5 seeds per task for each condition, averaged after merging nearby evaluation checkpoints. Per task results are in Figure A.5 in the Appendix. (b) Pessimism: normalized aggregate planner performance showing the effect of doing pessimism during acting as well, comparing pessimistic reanalyze against pessimistic acting and reanalyze across four HumanoidBench tasks, with 5 seeds per condition and task. Shaded regions denote 95% confidence intervals in both figures. Per task results are in Figure A.6 in the Appendix.

3.6. Conclusion

EfficientTDMPC improves BMPC by making the learned planner objective more reliable. By averaging return estimates across dynamics heads and rollout depths, and by using disagreement-based pessimism during reanalyze, EfficientTDMPC reduces the planner's sensitivity to single-model errors. Together with practical training improvements, these changes improve sample efficiency on HumanoidBench-Hard and Hard DMC while remaining competitive on Easy DMC. Our results suggest that in TD-MPC-style agents, improving the reliability of the objective optimized benefits the sample efficiency on the DMC and HumanoidBench-Hard benchmarks.

4

Joint Optimization of Action Sequences and Planning Horizon as Future Work

This chapter provides a detailed suggestion for future work on a general framework for uncertainty-aware model-based reinforcement learning. This framework treats the planning horizon itself as a decision variable that can be optimized. It trades off expected return and uncertainty in the same way as the choice of action sequence does. By jointly optimizing the action sequence and the planning horizon, the planner can find better tradeoffs between expected return and uncertainty. The proposed framework can be applied to any MPC-based planner that uses a learned model or value function to bootstrap the objective.

4.1. Related Work and Motivation

Explicit mean-variance tradeoff. Uncertainty-aware reinforcement learning works under the premise that when choosing actions, the agent should not only aim to maximize the expected return, but should also consider how certain the return estimate is. Uncertainty awareness has been used extensively in reinforcement learning, both for seeking out actions with uncertain outcomes for exploration [33] and avoiding uncertain actions for robustness. In model-free RL, disagreement between value heads in an ensemble has been successfully used as a measure of uncertainty. This uncertainty estimate is then often used as a bonus or penalty in the value target, policy objective, or both. Alternatively, a minimum or maximum over a subset of the ensemble is taken, which has the same effect.

With the rise of model-based RL, PETS proposed trajectory sampling as a method to estimate the uncertainty of model-based return estimates [8]. This involves rolling out an action sequence with multiple combinations of models and value functions, and then computing the variance of the resulting return estimates. This variance or standard deviation estimate, often called disagreement, can then be used as a proxy for uncertainty. Similarly to model-free RL, these uncertainty estimates can then be added to or subtracted from the return estimates [37, 44]. The resulting quantity can be interpreted as an upper or lower confidence bound on the return estimate. These bounds on the return estimate can then be used as simple value targets, but also as planning or policy objectives. Several model-based RL works such as PETS and CoPlanner [8, 44] also use uncertainty-aware model-based return estimates when evaluating action sequences.

Adaptive rollout horizon. In a seemingly separate line of work, several methods have proposed using uncertainty estimates to adapt the rollout horizon. Just as the choice of action sequence can trade off expected return and uncertainty, the choice of rollout depth does as well. When the actions rolled out come from the current policy, any rollout depth simply estimates the value of the current policy. Deeper rollouts simply add variance due to model error and policy stochasticity, and reduce bias slightly since they rely more on recent policy actions rather than a value bootstrap that is biased toward a slightly less recent policy. In this case, prior work has focused on reducing the variance of the estimates. Some works choose fixed rollout depths which empirically work well. Other options like TD-lambda [41], which was used by DreamerV3 [16], use a fixed weighted average of different rollout depths to reduce the variance. STEVE was the first to estimate the uncertainty at each rollout depth, and to optimally combine the different rollout depths to obtain the minimal variance [4]. None of these methods include an incentive to put more weight on longer rollout depths, except when doing so reduces the variance of the final estimate. The reason for this is that the actions rolled out are still from the policy, meaning the gain in expected return is often negligible. This is where policy or planner objectives come in. When the actions rolled out are optimized, instead of just sampled from the policy, deeper rollouts actually change the quantity being estimated. As a result, the increase in estimated return may be very significant. It is for this reason that methods like TD-MPC see greater empirical performance with planning depths of 3, rather than 1, which would be a more natural choice if the goal was just to reduce variance [18]. We see that the choice of planning depth controls a tradeoff between the mean and variance of the return estimate, just as the choice of action sequence does.

Joint optimization of action sequence and planning depth. We outline the following gap in the literature. Existing adaptive horizon selection methods have not been applied to planning objectives, nor are they suitable to use for planning objectives.

- **Adaptive horizon to minimize variance only.** Prior work adapts the rollout horizon to explicitly minimize the variance of the return estimate, ignoring the mean of the return estimate. This works well for rollouts of policy actions where the depth does not significantly change the expected return. However, this is very suboptimal in planning settings where deeper rollouts can significantly increase the expected return. In this case, the choice of planning depth should not only reduce the variance, but also increase the mean of the return estimate.
- **No joint optimization of action sequence and planning horizon.** To our knowledge, there exists no prior work in which the action sequences and planning horizon are optimized jointly, especially not by optimizing a convex combination over multiple planning horizons. We propose that this is suboptimal since the choice of action sequence and planning horizon are tightly dependent, and they both trade off the mean and variance of the return estimate.

4.2. Depth-Specific Planning Values

For a rollout depth τ, define the depth-τ state-action value of an action sequence as

$$Q_\tau^\pi(s_t, \mathbf{a}_{t:t+\tau-1}) = \sum_{k=0}^{\tau-1} \gamma^k r(s_{t+k}, a_{t+k}) + \gamma^\tau V^\pi(s_{t+\tau}). \tag{4.1}$$

The first τ actions are chosen explicitly, after which the return is bootstrapped with V^π. In the learned latent model, the corresponding estimate is

$$\hat{Q}_\tau^\pi(z_t, \mathbf{a}_{t:t+\tau-1}) = \sum_{k=0}^{\tau-1} \gamma^k R_\theta(\hat{z}_{t+k}, a_{t+k}) + \gamma^\tau V_\theta^\pi(\hat{z}_{t+\tau}), \qquad \hat{z}_{t+k+1} = f_\theta(\hat{z}_{t+k}, a_{t+k}). \tag{4.2}$$

This is the same type of model-based return estimate used in TD-MPC-style planners, but indexed by the rollout depth at which the value bootstrap is queried. For a fixed first action a_t, we define the τ optimized state-action value

$$\tilde{Q}^{\pi}_{\tau}(s_t, a_t) = \max_{\mathbf{a}_{t+1:t+\tau-1}} Q^{\pi}_{\tau}(s_t, \mathbf{a}_{t:t+\tau-1}). \tag{4.3}$$

This quantity represents the return that can be achieved by optimizing the first $\tau-1$ continuation actions after a_t. With exact dynamics, rewards, and values, deeper optimization monotonically increases the expected return:

$$\tilde{Q}^{\pi}_{1}(s_t, a_t) \leq \tilde{Q}^{\pi}_{\tau}(s_t, a_t) \leq \tilde{Q}^{\pi}_{\tau+1}(s_t, a_t) \leq \tilde{Q}^{\pi}_{\infty}(s_t, a_t) = Q^{*}(s_t, a_t). \tag{4.4}$$

The ideal planner would therefore use the longest available horizon, because a longer horizon replaces more of the policy bootstrap with optimized actions.

4.3. The Mean-Uncertainty Tradeoff

In reality the agent does not have access to the true Q^{π}_{τ}, but only to learned return estimates $\hat{Q}^{\pi}_{\tau}$. These estimates are not perfect and have some variance, denoted by $\sigma^2_{\hat{Q}^{\pi}_{\tau}}$, or σ^2_{τ} for brevity. Due to increasing rollout errors, the variance of the return estimates generally increases with the rollout depth τ:

$$\sigma^2_1 \leq \sigma^2_{\tau} \leq \sigma^2_{\tau+1} \leq \sigma^2_H. \tag{4.5}$$

As a result, choosing the planning depth involves a fundamental tradeoff between the expected return and the uncertainty of the return estimate. Deeper planning leads to higher expected return, but also to higher uncertainty.

4.4. Explicit Mean-Uncertainty Tradeoff with Lower Confidence Bounds

Prior work [8] in the uncertainty-aware RL literature has proposed to explicitly trade off return and uncertainty by maximizing a lower confidence bound on the return estimate. The agent estimates the uncertainty of the return estimate $\hat{\sigma}_{\tau}$, as well as the return estimate $\hat{Q}^{\pi}_{\tau}$ itself, and then uses the following planning objective:

$$J_{\tau,\beta}(\mathbf{a}) = \hat{Q}^{\pi}_{\tau}(\mathbf{a}) - \beta\hat{\sigma}_{\tau}(\mathbf{a}). \tag{4.6}$$

From this point onward, for brevity, we refer to dependence on $(z_t, \mathbf{a}_{t:t+\tau-1})$ as simply dependence on $\mathbf{a}$. The planner then finds the action sequence that maximizes this lower-confidence bound:

$$\mathbf{a}^{*} \in \arg\max_{\mathbf{a}} J_{\tau,\beta}(\mathbf{a}). \tag{4.7}$$

Here β controls the tradeoff between estimated return and uncertainty, and τ is a fixed planning depth.

4.5. Adaptive Horizon Selection

We propose that the choice of planning depth trades off expected return and uncertainty in the same way as the choice of action sequence. While previous work has generally chosen planning depth as a fixed hyperparameter, we propose that the planner should jointly optimize the action sequence and the planning depth. Instead of fixing τ in advance, the planner can choose the rollout depth that yields the

best lower-confidence bound:

$$\tau^* \in \arg \max_{\tau \in \{1,\dots,H\}} \max_{\mathbf{a}} \left[\hat{Q}^{\pi}_{\tau}(\mathbf{a}) - \beta \hat{\sigma}_{\tau}(\mathbf{a}) \right]. \tag{4.8}$$

Under unbiased return estimates and accurate uncertainty estimates, this objective selects the planning horizon that is optimal for the chosen lower-confidence-bound criterion, rather than using the same planning depth for all situations.

4.6. Convex Horizon Mixtures

The optimization can be made more flexible by allowing the planner to choose a convex combination of different depth return estimates instead of selecting a single depth. For $w \in \Delta_H$, where Δ_H is the probability simplex, define the true horizon-mixture value as

$$Q^{\pi}_{w}(\mathbf{a}) = \sum_{\tau=1}^{H} w_{\tau} Q^{\pi}_{\tau}(\mathbf{a}). \tag{4.9}$$

The corresponding learned estimate is the same convex combination of the depth-specific return estimates,

$$\hat{Q}^{\pi}_{w}(\mathbf{a}) = \sum_{\tau=1}^{H} w_{\tau} \hat{Q}^{\pi}_{\tau}(\mathbf{a}). \tag{4.10}$$

The variance of the mixture estimate is then simply

$$\sigma^2_{w}(\mathbf{a}) = \mathrm{Var}\left[\hat{Q}^{\pi}_{w}(\mathbf{a}) \right] = w^{\top} \Sigma(\mathbf{a}) w. \tag{4.11}$$

where $\Sigma(\mathbf{a})$ is the covariance matrix of the depth-specific return estimates. In practice, we estimate this covariance matrix as $\hat{\Sigma}(\mathbf{a}) \in \mathbb{R}^{H \times H}$. The diagonal entries are the estimated variances of the individual depths $\hat{\sigma}^2_{\tau}(\mathbf{a})$, while the off-diagonal entries are set to zero, corresponding to an approximation that ignores cross-depth correlations:

$$\hat{\Sigma}(\mathbf{a}) = \begin{bmatrix} \hat{\sigma}^2_1(\mathbf{a}) & 0 & \cdots & 0 \\ 0 & \hat{\sigma}^2_2(\mathbf{a}) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \hat{\sigma}^2_H(\mathbf{a}) \end{bmatrix}. \tag{4.12}$$

The estimated variance of the mixture is therefore

$$\hat{\sigma}^2_{w}(\mathbf{a}) = w^{\top} \hat{\Sigma}(\mathbf{a}) w. \tag{4.13}$$

The resulting uncertainty-aware horizon-mixture objective jointly optimizes the action sequence and the horizon weights:

$$\max_{\mathbf{a},\, w \in \Delta_H} \left[\hat{Q}^{\pi}_{w}(\mathbf{a}) - \beta \sqrt{w^{\top} \hat{\Sigma}(\mathbf{a}) w} \right]. \tag{4.14}$$

Assuming the different depth estimates are not perfectly correlated, this objective is able to reach a higher lower confidence bound on the return estimate compared to picking the single-depth objective, as proposed previously in Eq. (4.6).

4.7. Solving the Joint Optimization

The previous section introduces a joint optimization over action sequences and horizon weights. We propose a method to solve this efficiently by integrating a horizon-weight optimization step into the MPPI planner [48]. MPPI works by scoring proposed action sequences and then updating the action-sequence distribution to assign higher probability to better-scoring sequences. Standard MPPI scores a candidate action sequence with a fixed-depth return estimate such as $\hat{Q}_3^\pi(\mathbf{a})$. We propose to score each candidate with optimal horizon weights. Finding the optimal horizon weights for a candidate action sequence can be done relatively cheaply with a gradient-based optimizer, since the search space is just the simplex over H dimensions, and the objective is differentiable and cheap to evaluate.

Algorithm 4.1 gives pseudocode for MPPI planning with per-sequence horizon-weight optimization.

Algorithm 4.1 Simplified MPPI with per-sequence horizon-weight optimization

1. Initialize the action-sequence distribution $\mathcal{N}(\mu_a, \sigma_a^2)$ over trajectories of length H.
2. For each MPPI iteration:
   (a) Sample candidate action sequences $\mathbf{a}^{(n)}$ from the current distribution.
   (b) For each candidate action sequence $\mathbf{a}^{(n)}$:
      i. Estimate $\hat{Q}_\tau^\pi\left(\mathbf{a}^{(n)}\right)$ for all $\tau \in \{1, \ldots, H\}$ and $\hat{\Sigma}\left(\mathbf{a}^{(n)}\right)$.
      ii. **Find optimal horizon weights for this candidate using a gradient-based optimizer:**
      $$w^{(n),*} \in \arg\max_{w \in \Delta_H} \left[\hat{Q}_w^\pi\left(\mathbf{a}^{(n)}\right) - \beta\sqrt{w^\top \hat{\Sigma}\left(\mathbf{a}^{(n)}\right) w}\right].$$
      iii. Score the candidate with the optimized horizon weights,
      $$S^{(n)} = \hat{Q}_{w^{(n),*}}^\pi\left(\mathbf{a}^{(n)}\right) - \beta\sqrt{\left(w^{(n),*}\right)^\top \hat{\Sigma}\left(\mathbf{a}^{(n)}\right) w^{(n),*}}.$$
   (c) Use the scores $S^{(n)}$ to weight each candidate.
   (d) Update the action-sequence distribution mean and variance toward better-scoring candidates.
3. Execute the first action of the best or mean action sequence.

4.8. Sanity Check

We present a preliminary test of the proposed method. We trained the EfficientTDMPC agent on the reacher-hard task from DMC, and took a checkpoint of the agent's components after 50k environment steps. We then ran the proposed planner with per-sequence horizon-weight optimization on the learned model. Figure 4.1 shows the horizon weights throughout the MPPI iterations.

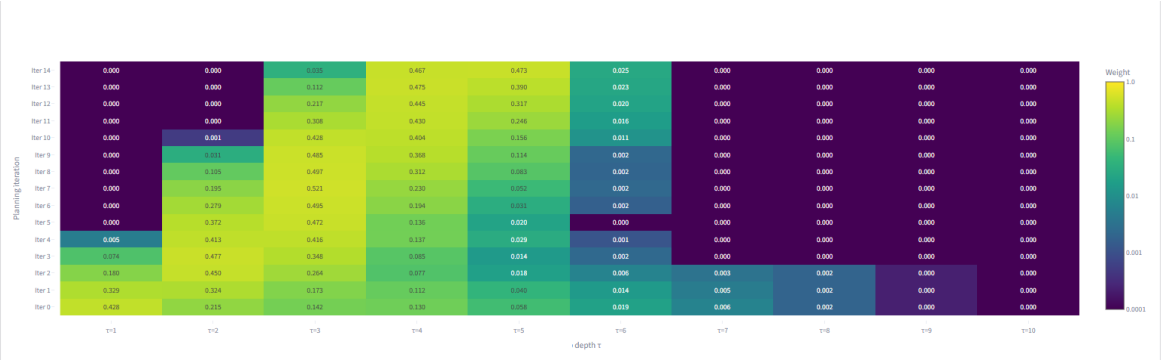


Figure 4.1: Horizon weights (color) throughout the MPPI iterations (y-axis) for each depth (x-axis), evaluated on a single reacher-hard state from a checkpoint at 50k environment steps.

We make several observations:

1. The horizon weights are not concentrated on a single depth, but rather spread out across multiple depths to reduce variance, as expected.
2. In early planning the horizon weights are mostly concentrated on the shorter horizons, but as the planner iterates, it focuses on deeper planning. This is also expected, since the planner is able to find better action sequences as it iterates, which makes deeper planning more effective.

Summary: Joint optimization of action sequence and planning horizon

We propose a direction for future work on a framework for uncertainty-aware model-based reinforcement learning in which the planning horizon is treated as a decision variable that can be optimized. By jointly optimizing the action sequence and a weighted combination of planning horizons, the planner can find better tradeoffs between expected return and uncertainty. The proposed framework can be applied to any MPC-based planner that uses a learned model or value function to bootstrap the objective. The proposed method requires a small change to the planner used in EfficientTDMPC, but may lead to significant performance improvements. The proposed solution aims to find the optimal planning depth for a given situation, instead of using the same planning depth for all situations. This includes different planning depths for different tasks, states, actions, points in training, and iterations of the planner.

5

Discussion and Conclusion

5.1. Discussion

This section discusses the results of the thesis and the insights that can be drawn from them in terms of the research questions. How should we change the model-based planner objective in TD-MPC-style agents to increase their sample efficiency? To what extent does this thesis provide insights on the general use of learned models by reinforcement learning agents to find actions that lead to high future reward?

5.1.1. Performance of EfficientTDMPC

One of the main results of this thesis is the sample efficiency that EfficientTDMPC was able to achieve on three well-known benchmarks for continuous control. EfficientTDMPC significantly outperformed the strong baseline BMPC on all three benchmarks by a large margin while using the same wall-clock time. Furthermore, the highest UTD setting of EfficientTDMPC has become the new state of the art on HumanoidBench-Hard and DMC hard, while matching the state of the art on DMC easy. It is also the only method to reach state-of-the-art performance on all three benchmarks. This is a strong result that demonstrates the effectiveness of the modifications proposed in this thesis.

5.1.2. Effects of individual contributions

While the performance of the final proposed system is strong, it is valuable to understand the contributions and interactions of the individual modifications that were made. The three main contributions proposed in this thesis were the dynamics ensemble, depth aggregation, and uncertainty-aware planning. Additionally, EfficientTDMPC made some auxiliary modifications, such as reducing the reanalyze compute and updating the replay buffer more frequently. Lastly, EfficientTDMPC showed that it benefits from higher UTD settings. The thesis contained experiments that ablate the individual effects of these modifications. Experiments on the individual effects of the dynamics ensemble and depth aggregation showed that while these components were more likely to have contributed rather than harmed performance, the number of seeds was not sufficient to conclude a statistically significant performance gain. Despite this, the final version of EfficientTDMPC did show significant improvements, even when tested on DMC with UTD 1. In this case the only modifications that were active were the dynamics ensemble, depth aggregation, and the auxiliary modifications. Furthermore, in this setting EfficientTDMPC benefited significantly more from higher UTD settings than the baseline. These improvements seen in this setting can be explained in one of the following ways. The first option is that the dynamics

ensemble and depth aggregation may have had some positive effect, but the number of seeds of the individual experiments was not sufficient to show a significant effect. The final experiments used more tasks and seeds, revealing statistically significant improvements all along. The second option is that one of the auxiliary modifications is responsible for the improvements. Our experiments do show that the reduced reanalyze compute, which is an auxiliary modification, actually improved performance weakly on HumanoidBench. While this seems counterintuitive, it may be responsible for a part of the improvements. The third option is that the modifications interacted positively with one another to yield more statistically significant improvements. It is unclear which of these options is the true explanation for the improvements. Uncertainty-aware planning, also named the pessimistic reanalyze, did show significant improvements on HumanoidBench, and is likely responsible for a large portion of the HumanoidBench improvements. We find that while pessimistic reanalyze is a tool that can significantly improve performance, it should not always be used, since it can also hurt performance on some tasks. It is important to note that the implementation of uncertainty-aware planning also required the use of a dynamics ensemble in order to estimate the uncertainty of the dynamics. For this reason, the dynamics ensemble may be crucial to the performance of EfficientTDMPC, perhaps not by reducing the error of the return estimates, but by enabling better uncertainty-aware planning. However, the thesis contains no experiments that ablate the effect of the dynamics ensemble on uncertainty-aware planning, so no definitive conclusions can be drawn on this.

5.1.3. Pessimism and Safety

Another important metric besides sample efficiency is the safety of the method interacting with the environment. This is especially important for real-world applications, where unsafe interactions can lead to damage to the system or its surroundings. One of the main contributions of this thesis is uncertainty-aware planning, which has been shown to significantly increase the sample efficiency of the method. In addition, it allows us to control the extent to which the agent seeks out states that are uncertain and new. Uncertainty-aware planning is therefore also a tool to control the agent's desire to take risks and explore new states and actions, which is an important component of the safety of the method.

5.1.4. Computational Cost and Hardware Effects

Another important metric of any reinforcement learning method is the computational cost. Especially in more realistic applications, modeling the world and planning with it can be computationally expensive, and the computational cost may be a bottleneck for the method. The ideas proposed in this thesis generally increase the computational cost of the method, since they require more model and value function evaluations. This is a large drawback. However, this thesis also provides an insight that saves compute, which allows the proposed EfficientTDMPC to have the same wall-clock training time as the baseline. This insight is that planning compute during acting in the environment should not be the same as planning compute during reanalyze, that is, when refreshing policy targets. Our experiments show that reducing the number of planning particles during reanalyze can actually increase performance, while reducing the number of reanalyze FLOPS by 8x. Our intuitive explanation for this is that the original planner was designed to act in the environment. When acting in the environment, we want to take actions that try to exploit the errors in our estimates, because this is how we collect new data to get those errors fixed. For this reason, expensive planning is justified, since it better exploits those errors. However, when we do reanalyze planning, we are defining the actions that we want our policy to follow. We do not want our policy to take exploitative actions, since this will destabilize value functions. As a result, the method may even benefit from cheaper reanalyze planning. Despite saving compute on reanalyze, our method does still require more FLOPS than the baseline, primarily due to the planning compute when acting in the environment. We predict that our wall-clock training time is equal in the

UTD 1 case due to two reasons. First, our implementation differs from that of the baseline, and it may be more optimized. Second, we test on a larger A100 GPU, whereas the baseline was tested on a slightly smaller RTX 3090 GPU. It may be that our extra FLOPS saturate the larger GPU better and are therefore not converted to extra wall-clock time.

5.1.5. Broader Impact

The broader goal of this thesis is to gain insights that extend into more realistic physical intelligence systems. Such a setting deals with more complicated environments that have high-dimensional observations and only partially observable states. As stated in the introduction, to do reinforcement learning in such environments, one ongoing plan is to pretrain a world model and policy via imitation, and then fine-tune them with reinforcement learning. The question becomes: to what extent will the insights from this thesis extend to such systems? While a definitive answer to this question will require experiments that are left to future work, we can speculate on properties that will limit the transferability of the insights, as well as which insights may be relevant for such realistic systems. One big bottleneck that limits the transferability of the insights in this thesis to such a setting is that the TD-MPC framework trains a value-equivalent world model. This means that the latent state does not describe the full observation, but rather only what matters for the task, and the dynamics are trained to predict the value accurately, not just the dynamics. While this is a powerful concept, it does not align well with the idea of pretrained world models. Pretrained world models should, by definition, model the full environment. While this does not limit the transferability of the ideas in this thesis, it may limit the extent to which the empirical benefits of the ideas in this thesis transfer to such a setting. For example, the dynamics ensemble may be less effective when the dynamics are pretrained. Without further experimentation, it is unclear to what extent the validity of the insights in this thesis will transfer to more realistic physical intelligence systems.

5.1.6. Future work

Future work should first strengthen the causal evidence for the individual components of EfficientTDMPC. In particular, more seeds and broader factorial ablations are needed to separate the effects of the dynamics ensemble, depth aggregation, uncertainty-aware planning, reduced reanalyze compute, replay-buffer updates, and higher UTD settings. This would clarify whether the gains of the full method are caused by specific components, by interactions between them, or by auxiliary implementation changes. Future work should also experiment more with the dynamics ensemble. Dynamics ensembles have been shown to be very effective in other model-based RL works, but EfficientTDMPC has been the first work to apply them to value-equivalent world models. Further experiments should be done to understand the effects of the dynamics ensemble on value-equivalent world models. Second, the effect of the dynamics ensemble on uncertainty-aware planning should be studied. The aim of this would be to understand whether the uncertainty-aware planning significantly benefits from uncertainty estimates that take the uncertainty in the dynamics into account, or whether the uncertainty from the value head disagreement is sufficient. Another important direction is to evaluate the method on a broader set of environments. The current experiments already show that pessimism is beneficial on HumanoidBench but can hurt on DMC, which suggests that the value of uncertainty-aware planning may be strongly domain dependent. Testing more benchmarks would help identify which task properties make pessimism or higher UTD ratios useful. An extension of this is moving towards benchmarks and baselines that deal with more realistic scenarios. Future work should see to what extent the ideas proposed in this thesis hold up under these different conditions. Lastly, Chapter 4 has proposed a detailed direction for future work that solves problems with the existing horizon aggregation idea, while combining it into a unified framework with uncertainty-aware planning. The idea is to jointly optimize a convex combination of planning horizons, as well as the action sequence, in order to maximize an uncertainty-aware planning objective.

5.2. Conclusion

This thesis studied how learned models should be used by reinforcement-learning agents to select actions that lead to high future reward. The broader motivation is that future physical-intelligence systems will likely depend on learned world models, policy priors, and reinforcement learning. In such systems, the agent will not act by directly knowing the true consequences of its actions. Instead, it will choose actions by optimizing predictions made by learned models. This makes the reliability of the learned planning objective a central problem. The thesis addressed this problem in a narrower setting: TD-MPC-style agents for sample-efficient continuous control. These agents learn a latent dynamics model and use it inside an MPC planner. This can make action selection much stronger than directly sampling from the policy, but it also introduces a failure mode. The planner optimizes a learned return estimate, not the true environment return. If the learned dynamics, reward model, or value function contains errors, a powerful planner may exploit those errors and select actions that look good under the model but perform poorly in the environment. The main answer provided by this thesis is that the MPC objective should not rely on a single learned prediction of the future. Instead, the planning objective should be made more robust by aggregating information across multiple model-based estimates and by accounting for uncertainty. EfficientTDMPC implements this idea by averaging return estimates across dynamics heads and rollout depths, and by using disagreement-based pessimism during reanalyze. These changes are designed to make the planner less sensitive to individual model errors and less likely to produce over-optimistic policy targets from uncertain imagined trajectories.

Empirically, EfficientTDMPC achieves strong sample-efficiency improvements over BMPC. On the tested continuous-control benchmarks, EfficientTDMPC significantly outperforms BMPC while maintaining comparable wall-clock training time. At its strongest setting, it reaches a new state of the art on HumanoidBench-Hard and DMC Hard, while matching state-of-the-art performance on DMC Easy. It is also the only evaluated method among the baselines to have reached state-of-the-art-level performance across all three benchmark groups. This indicates that improving the reliability of the learned MPC objective can lead to substantial gains in sample-efficient model-based reinforcement learning. At the same time, the individual role of each component is not fully resolved. The clearest isolated result is that uncertainty-aware planning through pessimistic reanalyze is able to yield significant improvements on HumanoidBench. Increasing the UTD ratio also provides a clear benefit for EfficientTDMPC. However, the isolated effects of the dynamics ensemble and depth aggregation are less certain. The ablations do not provide enough statistical evidence to conclude that either component independently improves performance. Nevertheless, the full system produces significant improvements, even when the scaled UTD and pessimistic reanalyze are not active. This suggests that the gains may come from weaker individual effects that become visible only when combined, from interactions between components, or from auxiliary implementation changes such as reduced reanalyze compute and more frequent replay-buffer updates. Furthermore, the uncertainty-aware planning component depends on the dynamics ensemble to estimate uncertainty in the dynamics, so the ensemble may be crucial to the performance of EfficientTDMPC simply because it enables better uncertainty-aware planning.

Overall, the thesis supports the view that sample-efficient model-based reinforcement learning depends not only on learning better components, but also on using imperfect components more carefully when planning. EfficientTDMPC shows that aggregating model-based return estimates and incorporating uncertainty into planning can improve this objective and push the sample-efficiency frontier on challenging continuous-control benchmarks. This provides evidence that planning objectives that aim to be more robust to component errors are a promising direction for future reinforcement-learning systems, especially in settings where interaction with the environment is expensive, risky, or slow.

References

[1] Richard Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[2] Tom B. Brown et al. "Language Models are Few-Shot Learners". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 33. 2020.

[3] Cameron B. Browne et al. "A Survey of Monte Carlo Tree Search Methods". In: *IEEE Transactions on Computational Intelligence and AI in Games* 4.1 (2012), pp. 1–43.

[4] Jacob Buckman et al. "Sample-Efficient Reinforcement Learning with Stochastic Ensemble Value Expansion". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2018.

[5] Wei-Di Chang et al. *The Surprising Difficulty of Search in Model-Based Reinforcement Learning*. 2026. arXiv: 2601.21306 [cs.LG]. URL: https://arxiv.org/abs/2601.21306.

[6] Xinyue Chen et al. "Randomized Ensembled Double Q-Learning: Learning Fast Without a Model". In: *International Conference on Learning Representations (ICLR)*. 2021.

[7] Paul F. Christiano et al. "Deep Reinforcement Learning from Human Preferences". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2017.

[8] Kurtland Chua et al. "Deep Reinforcement Learning in a Handful of Trials using Probabilistic Ensemble Trajectory Sampling". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2018.

[9] Jim Fan. *Robotics' End Game*. Training Data podcast / talk. Non-peer-reviewed source; use only for attribution of the "Great Parallel" framing or figure source. 2026.

[10] Scott Fujimoto, Herke Hoof, and David Meger. "Addressing function approximation error in actor-critic methods". In: *International conference on machine learning*. PMLR. 2018, pp. 1587–1596.

[11] Christopher Grimm et al. "The Value Equivalence Principle for Model-Based Reinforcement Learning". In: *Advances in Neural Information Processing Systems*. 2020.

[12] Daya Guo, Dejian Yang, Haowei Zhang, et al. "DeepSeek-R1 Incentivizes Reasoning in LLMs Through Reinforcement Learning". In: *Nature* 645 (2025), pp. 633–638. DOI: 10.1038/s41586-025-09422-z. URL: https://doi.org/10.1038/s41586-025-09422-z.

[13] David Ha and Jürgen Schmidhuber. "World Models". In: *arXiv preprint arXiv:1803.10122* (2018).

[14] Tuomas Haarnoja et al. "Soft Actor-Critic Algorithms and Applications". In: *arXiv preprint arXiv:1812.05905* (2019).

[15] Danijar Hafner et al. "Dream to Control: Learning Behaviors by Latent Imagination". In: *International Conference on Learning Representations (ICLR)*. 2020.

[16] Danijar Hafner et al. "Mastering diverse control tasks through world models". In: *Nature* 640.8059 (Apr. 2025), pp. 647–653. DOI: 10.1038/s41586-025-08744-2. URL: https://www.nature.com/articles/s41586-025-08744-2.

[17] Nicklas Hansen. *tdmpc2 issue #44: Incorrect entropy due to tanh squashing*. GitHub issue: https://github.com/nicklashansen/tdmpc2/issues/44. 2024.

[18] Nicklas Hansen, Hao Su, and Xiaolong Wang. "TD-MPC2: Scalable, Robust World Models for Continuous Control". In: *International Conference on Learning Representations (ICLR)*. 2024.

[19] Nicklas Hansen, Xiaolong Wang, and Hao Su. "Temporal Difference Learning for Model Predictive Control". In: *International Conference on Machine Learning (ICML)*. 2022.

[20] Jordan Hoffmann et al. "Training Compute-Optimal Large Language Models". In: *arXiv preprint arXiv:2203.15556* (2022).

[21] Thomas Hubert, Rishi Mehta, Laurent Sartran, et al. "Olympiad-Level Formal Mathematical Reasoning with Reinforcement Learning". In: *Nature* 651 (2026), pp. 607–613. DOI: 10.1038/s41586-025-09833-y. URL: https://doi.org/10.1038/s41586-025-09833-y.

[22] Michael Janner et al. "When to Trust Your Model: Model-Based Policy Optimization". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2019.

[23] Jared Kaplan et al. "Scaling Laws for Neural Language Models". In: *arXiv preprint arXiv:2001.08361* (2020).

[24] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 25. 2012.

[25] Yann LeCun. *A Path Towards Autonomous Machine Intelligence*. OpenReview position paper. Version 0.9.2, 2022-06-27. 2022. URL: https://openreview.net/forum?id=BZ5a1r-kVsf.

[26] Haotian Lin et al. *TD-M(PC)^2: Improving Temporal Difference MPC Through Policy Constraint*. 2025. arXiv: 2502.03550 [cs.LG]. URL: https://arxiv.org/abs/2502.03550.

[27] Cristian Meo et al. *Masked Generative Priors Improve World Models Sequence Modelling Capabilities*. 2025. arXiv: 2410.07836 [cs.LG]. URL: https://arxiv.org/abs/2410.07836.

[28] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), pp. 529–533.

[29] OpenAI. *Learning to Reason with LLMs*. OpenAI blog. Accessed 2026-06-13. 2024. URL: https://openai.com/index/learning-to-reason-with-llms/.

[30] OpenAI. "OpenAI Five". In: *OpenAI Blog* (2019). URL: https://openai.com/index/openai-five/.

[31] Yaniv Oren et al. "Twice Sequential Monte Carlo for Tree Search". In: *arXiv preprint arXiv:2511.14220* (2025). DOI: 10.48550/arXiv.2511.14220.

[32] Yaniv Oren et al. "Value Improved Actor Critic Algorithms". In: *arXiv preprint arXiv:2406.01423* (2024). DOI: 10.48550/arXiv.2406.01423.

[33] Ian Osband et al. "Deep Exploration via Bootstrapped DQN". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2016.

[34] Long Ouyang et al. "Training Language Models to Follow Instructions with Human Feedback". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 35. 2022.

[35] Paria Rashidinejad et al. *Bridging Offline Reinforcement Learning and Imitation Learning: A Tale of Pessimism*. 2023. arXiv: 2103.12021 [cs.LG]. URL: https://arxiv.org/abs/2103.12021.

[36] Julian Schrittwieser et al. "Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model". In: *Nature* 588 (2020), pp. 604–609.

[37] Ramanan Sekar et al. "Planning to Explore via Self-Supervised World Models". In: *International Conference on Machine Learning (ICML)*. 2020.

[38] Carmelo Sferrazza et al. "HumanoidBench: Simulated Humanoid Benchmark for Whole-Body Locomotion and Manipulation". In: *Robotics: Science and Systems (RSS)*. 2024.

[39] David Silver et al. "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm". In: *arXiv preprint arXiv:1712.01815* (2017).

[40] Richard S. Sutton. "Learning to Predict by the Methods of Temporal Differences". In: *Machine Learning* 3.1 (1988), pp. 9–44.

[41] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. 2nd ed. MIT Press, 2018.

[42] Yuval Tassa et al. "DeepMind Control Suite". In: *arXiv preprint arXiv:1801.00690* (2018).

[43] Oriol Vinyals et al. "Grandmaster level in StarCraft II using multi-agent reinforcement learning". In: *Nature* 575.7782 (2019), pp. 350–354.

[44] Xiyao Wang et al. "COPlanner: Plan to Roll Out Conservatively but to Explore Optimistically for Model-Based RL". In: *Proceedings of the Twelfth International Conference on Learning Representations*. 2024. URL: https://openreview.net/forum?id=jnFcKjtUPN.

[45] Yiping Wang et al. "Reinforcement Learning for Reasoning in Large Language Models with One Training Example". In: *arXiv preprint arXiv:2504.20571* (2025). URL: https://arxiv.org/abs/2504.20571.

[46] Yuhang Wang et al. "Bootstrapped Model Predictive Control". In: *The Thirteenth International Conference on Learning Representations*. 2025.

[47] Xumeng Wen et al. "Reinforcement Learning with Verifiable Rewards Implicitly Incentivizes Correct Reasoning in Base LLMs". In: *arXiv preprint arXiv:2506.14245* (2025). URL: https://arxiv.org/abs/2506.14245.

[48] Grady Williams et al. "Information Theoretic MPC: Theory and Applications to Autonomous Driving". In: *IEEE Transactions on Robotics* 34.6 (2018), pp. 1603–1622. DOI: 10.1109/TRO.2018.2865891.

[49] Tianhe Yu et al. "MOPO: Model-based Offline Policy Optimization". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2020.

[50] Guojian Zhan et al. "Bootstrap Off-policy with World Model". In: *Advances in Neural Information Processing Systems (NeurIPS)* 38 (2025).

A

Supplementary Material for the EfficientTDMPC Chapter

A.1. Implementation details

A.1.1. Hyperparameters

Table A.1 compares the final EfficientTDMPC / UTD4 configuration used in the paper against base BMPC. Entries in the EfficientTDMPC / UTD4 column are boldfaced when they differ from BMPC.

Table A.1: EfficientTDMPC / UTD4 versus BMPC hyperparameters. Bold entries in the EfficientTDMPC / UTD4 column denote values that differ from BMPC.

Category	Parameter	EfficientTDMPC UTD4	BMPC
Architecture	Dynamics heads	**4**	1
	Latent dim	512	512
	MLP hidden dim	512	512
	MLP layers	2	2
	Encoder dim / layers	256 / 2	256 / 2
	SimNorm dim	8	8
	Value heads	2	2
	Reward heads	1	1
Training	Batch size	256	256
	Learning rate	3×10^{-4}	3×10^{-4}
	Encoder LR scale	0.3	0.3
	Grad clip norm	20	20
	EMA τ	0.01	0.01
	UTD schedule	**UTD4**	UTD1
	Model / Value / Policy updates per env step	**4 / 4 / 4**	1 / 1 / 1
	Reanalyze interval	**2**	10
Acting planner (MPPI)	Horizon	**6**	3
	Horizon aggregation	**mean**	final
	Iterations	6 (+2 if $\|A\| > 20$)	6 (+2 if $\|A\| > 20$)
	Samples	512	512
	Elites	64	64
	Policy trajectories	24	24
	Temperature	0.5	0.5
	Min / Max std	0.05 / 2.0	0.05 / 2.0
Reanalyze planner (MPPI)	Horizon	3	3
	Horizon aggregation	final	final
	Iterations	6 (+2 if $\|A\| > 20$)	6 (+2 if $\|A\| > 20$)
	Samples	**64**	512
	Elites	**8**	64
	Policy trajectories	**3**	24
	Temperature	0.5	0.5
	Min / Max std	0.05 / 2.0	0.05 / 2.0
	Batch size	20	20
	Pessimism coeff.	$\boldsymbol{\beta} = 10$ (0 on DMC)	β=0
Policy	Log std bounds	$[-3, 1]$	$[-3, 1]$
	Entropy coeff	10^{-4}	10^{-4}
	Optimization	distillation	distillation
Discount	$\gamma_{\min}$ / $\gamma_{\max}$	0.95 / 0.995	0.95 / 0.995
	ρ	0.5	0.5
	Value bins / range	101 / $[-10, 10]$	101 / $[-10, 10]$
Replay	Buffer update interval	**every step**	every episode

The table above should be read as the paper's final EfficientTDMPC / UTD4 configuration against

the BMPC baseline rather than as a literal identity statement for every planner call. EfficientTDMPC / UTD4 uses 4 model, value, and policy updates per environment step and a reanalyze interval of 2, compared with BMPC's UTD1 schedule and reanalyze interval of 10. The acting planner uses horizon 6 with mean aggregation, while the reanalyze planner is configured separately with horizon 3, terminal-only aggregation, and the smaller 64 / 8 / 3 sample, elite, and policy budget shown above. In code, both planners receive an additional two iterations automatically when the action dimension is greater than 20. Reanalyze uses batch size 20 for both methods, applies the paper coefficient β=10 on the high-action-dimension tasks, and uses β=0 on DMC.

A.1.2. Generating performance figures

This section describes how we generate the performance figures used in the paper, including per-task learning curves, normalized learning curves, aggregate normalized learning curves, area under the curve (AUC), and aggregate normalized AUC over the three benchmarks.

Per-task learning curves. For a task i with N_i seeds, let $R_{i,j,t}$ denote the evaluation return of seed j at evaluation step t. We compute the per-task mean return as

$$\hat{\mu}_{i,t} = \frac{1}{N_i} \sum_{j=1}^{N_i} R_{i,j,t}.$$

The across-seed standard deviation is

$$\hat{\sigma}_{i,t} = \sqrt{\frac{1}{N_i - 1} \sum_{j=1}^{N_i} \left(R_{i,j,t} - \hat{\mu}_{i,t}\right)^2}.$$

The standard error of the task mean is then

$$\hat{s}_{i,t} = \frac{\hat{\sigma}_{i,t}}{\sqrt{N_i}}.$$

Unless stated otherwise, shaded regions for per-task curves show the approximate 95% confidence interval

$$\hat{\mu}_{i,t} \pm 1.96 \hat{s}_{i,t}.$$

Normalized learning curves. For aggregate comparisons across tasks, we normalize each task by the final performance of BMPC on that task. Specifically, let c_i denote the final mean BMPC return on task i. The normalized mean return of method m on task i is

$$\tilde{\mu}^m_{i,t} = \frac{\hat{\mu}^m_{i,t}}{c_i}.$$

The corresponding normalized standard error is

$$\tilde{s}^m_{i,t} = \frac{\hat{s}^m_{i,t}}{c_i}.$$

In this calculation, the normalization constant c_i is treated as fixed. This means that the plotted confidence intervals capture uncertainty from finite seeds for the evaluated method, but do not include uncertainty in the BMPC normalization constant itself.

Aggregate normalized learning curves. For a benchmark with T tasks, we compute the aggregate normalized mean return as the average over task-level normalized means:

$$\tilde{\mu}_t^m = \frac{1}{T} \sum_{i=1}^{T} \tilde{\mu}_{i,t}^m.$$

To compute uncertainty, we view each task-level mean estimate as an independent Gaussian random variable,

$$X_{i,t}^m \sim \mathcal{N}\left(\tilde{\mu}_{i,t}^m, (\tilde{s}_{i,t}^m)^2\right),$$

where $\tilde{s}_{i,t}^m$ is the standard error of the normalized task mean. The aggregate benchmark estimate is

$$Y_t^m = \frac{1}{T} \sum_{i=1}^{T} X_{i,t}^m.$$

Therefore,

$$\mathbb{E}[Y_t^m] = \frac{1}{T} \sum_{i=1}^{T} \tilde{\mu}_{i,t}^m,$$

and, assuming independence across task-level seed estimates,

$$\mathrm{Var}(Y_t^m) = \frac{1}{T^2} \sum_{i=1}^{T} (\tilde{s}_{i,t}^m)^2.$$

The aggregate standard error is therefore

$$\tilde{s}_t^m = \frac{1}{T} \sqrt{\sum_{i=1}^{T} (\tilde{s}_{i,t}^m)^2}.$$

The shaded region for aggregate normalized curves shows

$$\tilde{\mu}_t^m \pm 1.96 \tilde{s}_t^m.$$

This interval represents the uncertainty in the mean value over all tasks.

Area under the curve. When computing the AUC of a learning curve, we first interpolate all compared learning curves to a shared evaluation grid. We then compute the AUC using numerical integration over this shared grid. This ensures that methods evaluated at different checkpoints are compared on the same set of environment-step values.

Aggregate normalized AUC over three benchmarks. To compute Figure 3.1, we first compute the aggregate normalized learning curve separately for Easy DMC, Hard DMC, and HumanoidBench-Hard. We then compute the AUC of each aggregate normalized learning curve. The final aggregate normalized AUC is the average of these three benchmark-level AUC values. Lastly, we normalize this quantity by the strongest compared baseline in the same metric (MRS.Q), which makes the value directly interpretable relative to the best baseline aggregate.

A.1.3. Baseline details

We compare EfficientTDMPC against recent model-based and model-free baselines for continuous control. Whenever possible, we use official result files released by the authors. When official results are

not available for a specific benchmark setting, we run the official implementation with the benchmark-specific configuration described below.

SAC. Soft Actor-Critic (SAC) is an off-policy model-free actor-critic baseline. We use the SAC results provided in the official BMPC repository: `https://github.com/wertyuilife2/bmpc`.

DreamerV3. DreamerV3 is a latent world-model agent trained with imagination-based policy optimization. We use the DreamerV3 results provided in the official BMPC repository: `https://github.com/wertyuilife2/bmpc`.

TD-MPC2. TD-MPC2 is the main TD-MPC-family baseline preceding BMPC, using latent dynamics, value bootstrapping, and MPPI planning. We use the TD-MPC2 results provided in the official BMPC repository: `https://github.com/wertyuilife2/bmpc`.

BMPC. Bootstrapped Model Predictive Control (BMPC) is the primary baseline for our method. EfficientTDMPC is built on top of BMPC. We use the official BMPC implementation and released result files from: `https://github.com/wertyuilife2/bmpc`.

BOOM. BOOM is a model-based RL method that bootstraps an off-policy policy using world-model planning. We use the released BOOM result files from the official BOOM repository: `https://github.com/molumitu/BOOM_MBRL`.

BRC. Bigger, Regularized, Categorical (BRC) is a strong model-free baseline based on high-capacity categorical value functions. We use the released result files from the official BRC repository: `https://github.com/naumix/BiggerRegularizedCategorical/tree/main/results`.

SimBaV2. SimBaV2 is a strong model-free baseline based on hyperspherical normalization. For DMC, we use the released result files from the official SimBaV2 repository: `https://github.com/DAVIAN-Robotics/SimbaV2/tree/master/results`. For HumanoidBench-Hard, we run the official SimBaV2 implementation and set the action repeat to 1, matching the HumanoidBench protocol.

MRS.Q. MRS.Q is a recent model-free baseline that studies pessimism and search in model-based reinforcement learning settings. We use the released result files from the official MRS.Q repository: `https://github.com/facebookresearch/MRSQ/tree/main/results`.

PPO. Proximal Policy Optimization (PPO) is included as an additional model-free baseline on HumanoidBench-Hard. We use the PPO results from the official HumanoidBench codebase: `https://github.com/carlosferrazza/humanoid-bench`.

A.1.4. UTD runtime comparison implementation details

We compare the wallclock time of EfficientTDMPC with different UTD ratios against the official BMPC implementation. We compare the time it takes to train for 200k environment steps on a single NVIDIA A100 GPU. We exclude compilation and evaluation time.

A.2. Per task results

A.2.1. Per-task learning curves humanoidBench

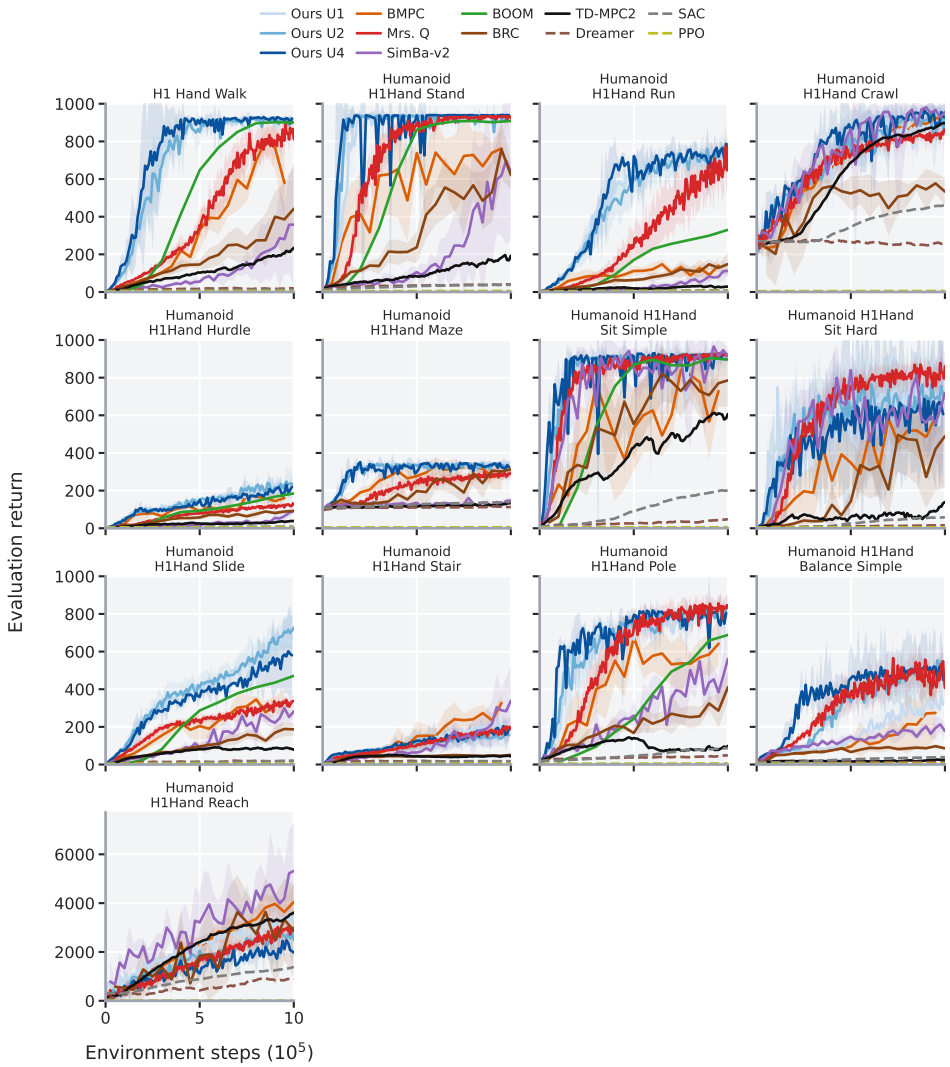


Figure A.1: Per-task results: Humanoid-Bench (300k decision steps / 600k environment steps). Evaluation return on all 13 Humanoid-Bench tasks with all available baselines. The 7 tasks for which BOOM has data are used for the aggregate in the main paper, but we include all 13 tasks here for completeness.

A.2.2. Per-task learning curves Hard DMC

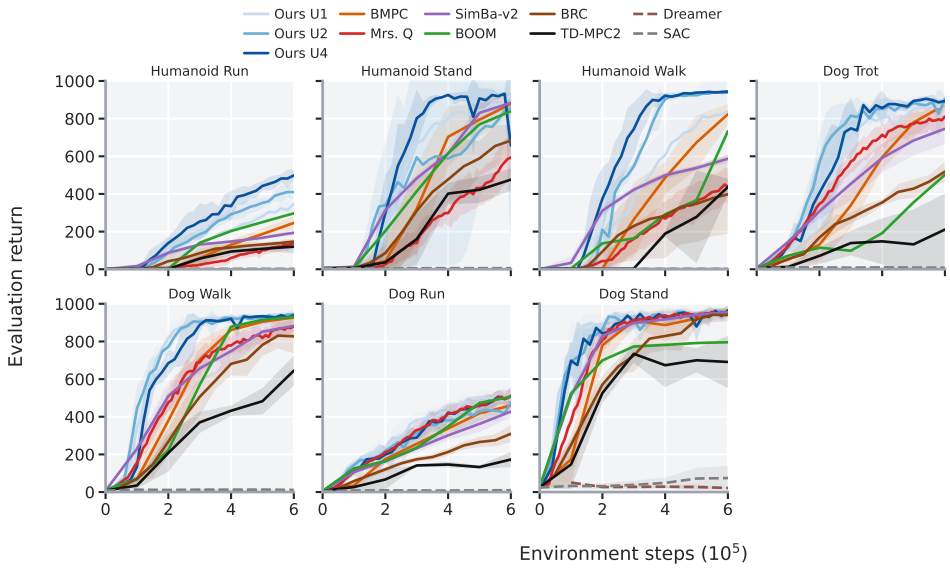


Figure A.2: Per-task results: Hard DMC (300k decision steps / 600k environment steps). Evaluation return on all 7 hard DMC tasks with all available baselines.

A.2.3. Per-task learning curves Easy DMC

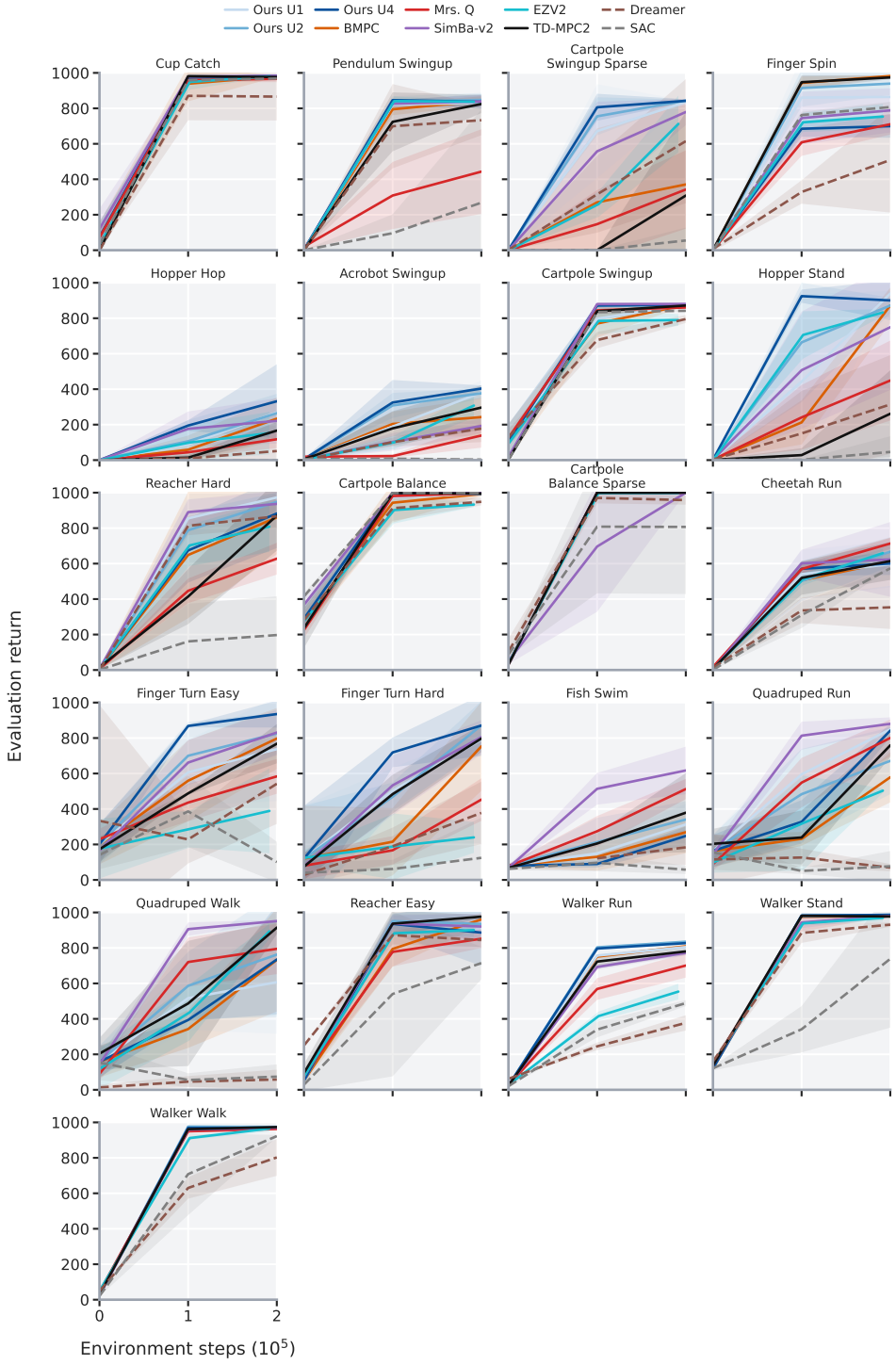


Figure A.3: Per-task results: Easy DMC (100k decision steps). Evaluation return on all 21 easy DMC tasks with all available baselines.

A.2.4. Per-task component ablations

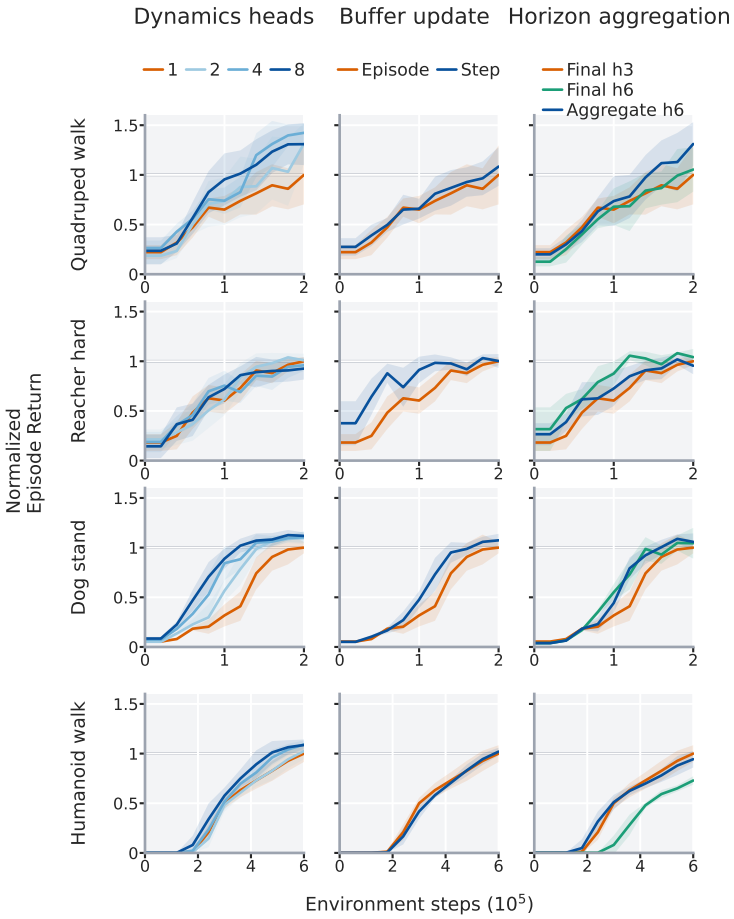


Figure A.4: Per-task component ablations. Learning curves for the isolated component ablations on quadruped-walk, reacher-hard, dog-stand, and humanoid-walk. Columns show the dynamics-ensemble, replay-buffer-update, and horizon-aggregation ablations; shaded regions denote 95% confidence intervals over available seeds.

A.2.5. Reduced reanalyze compute

We experiment with the effect of reduced reanalyze compute on EfficientTDMPC by using a configuration that cuts the number of MPPI samples from 512 to 64, the number of elites from 64 to 8, and the number of policy trajectories from 24 to 3. Because reducing the particle budget should be most harmful in large action/search spaces, we ablate this change on HumanoidBench-Hard tasks with 61-dimensional actions. Figure A.5 shows that decreasing the reanalyze budget does not degrade performance on any of the 4 tasks ablated.

Figure A.5: Reduced reanalyze compute does not degrade performance much. The per-task planner and policy learning curves show that training BMPC with 8x fewer reanalyze particles does not noticeably degrade performance on four HumanoidBench tasks with 61 action dimensions. The accompanying compact aggregate summarizes the normalized planner performance over the same four tasks. Shaded regions denote 95% confidence intervals over available seeds.

A.2.6. HumanoidBench pessimism scope

EfficientTDMPC applies pessimism only during reanalyze. We ablate the effect of applying pessimism when acting in the environment as well. On 4 tasks from HumanoidBench, we compare the reanalyze-only pessimism setting against pessimism during all planning. We find that additionally applying pessimism during acting does not improve performance. We use the same pessimism coefficient β=10 for both conditions.

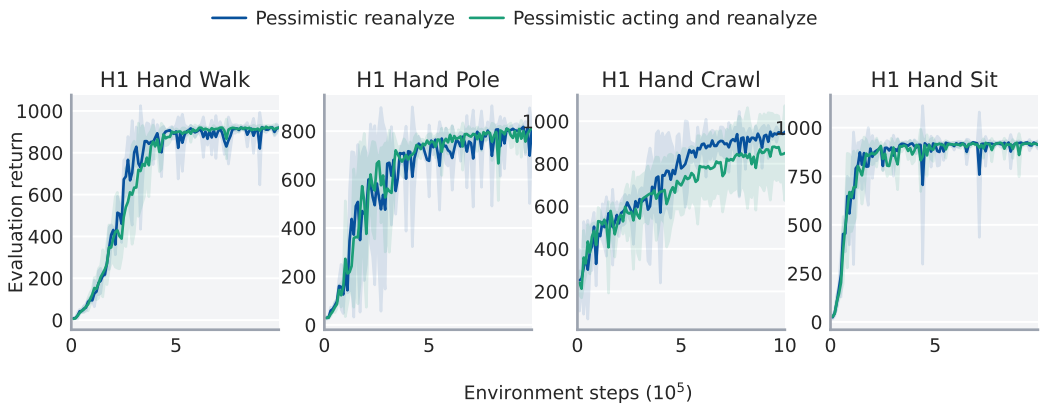


Figure A.6: HumanoidBench pessimism scope. Left: per-task evaluation return versus environment interactions on four HumanoidBench core tasks when pessimism is applied during reanalyze only versus during training-time planning, evaluation-time planning, and reanalyze. Right: the normalized aggregate over the same four tasks. Shaded regions denote 95% confidence intervals over available seeds.

A.3. Qualitative analysis

A.3.1. Planner cross-scoring quantities

We study the effect of averaging the planner objective over multiple ensemble members. On four representative tasks, we take checkpoints of the agent and replay buffer during training and run planning on a subset of replay-buffer states. We define three estimates of return. The first uses a single value head and a single dynamics head. The second averages over the two value heads and four dynamics heads, which is the EfficientTDMPC acting planner objective. The third return estimate approximates the true return by using a three-step rollout in the true environment simulator and then bootstraps with the learned value heads.

We then plan using both the single-head return estimate and the ensemble-averaged return estimate. To match the reduced-budget setting used by EfficientTDMPC reanalyze, these appendix experiments use a six-iteration MPPI planner with 64 samples, eight elites, three policy trajectories, and temperature 0.5. We score the resulting action sequences under all 3 return estimates. For each action sequence we compare its estimated return with the estimated return of the policy network, then extrapolate the resulting three-step difference to a 500-step episode to obtain the ΔR quantity plotted in Figure A.7. This ΔR value estimates how much the planner is predicted to outperform the mean policy action from the same state. The plotted means average over 512 replay states, and the error bars show the standard error across those states.

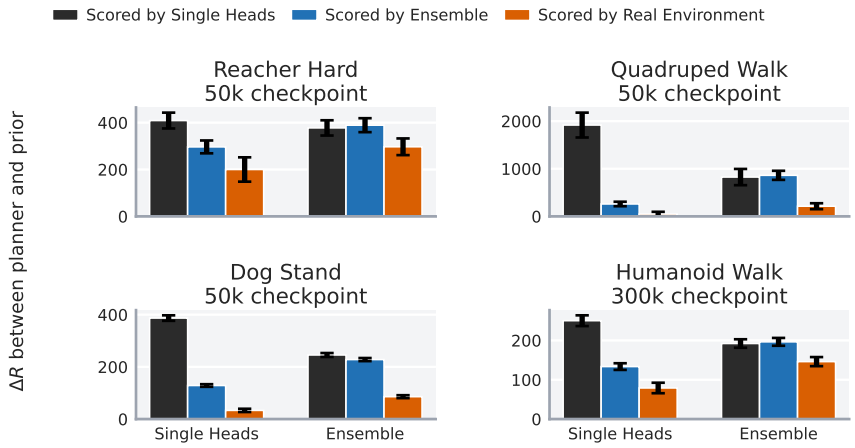


Figure A.7: Ensemble averaging of return estimate reduces single-head planner exploitation. The plotted ΔR quantity estimates how much each planner is predicted to outperform the policy action from the same state. Bars show means over 512 replay states and error bars show standard errors over states.

We find that the action sequences maximized under the single-head return estimate are given a very high return under that same single-head estimate, however, when evaluated under the real environment dynamics, the return is much lower and often worse than the policy actions. In contrast, the action sequences optimized under the ensemble-averaged return estimate are more modestly estimated to outperform the policy actions under their own return estimate, but they are also more accurately estimated to outperform the policy actions under the real environment dynamics. This suggests that the ensemble-averaged return objective is more robust to model bias and prevents the planner from exploiting errors in a single value and dynamics head.

A.3.2. Latent trajectory gallery

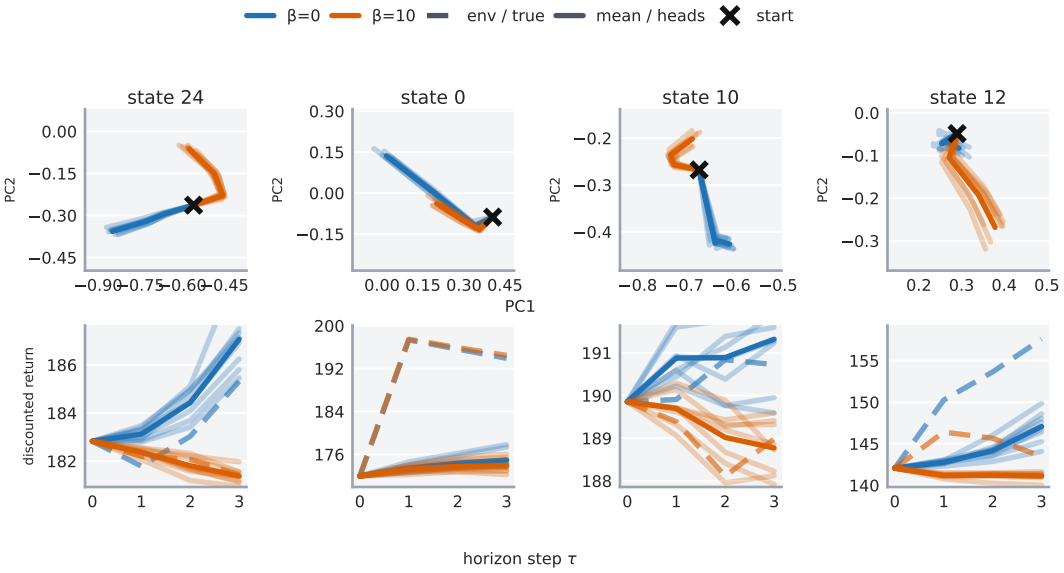


Figure A.8: Latent trajectory gallery on reacher-hard at 50k (four states). Blue denotes actions from the optimistic planner ($\beta = 0$) and orange from the pessimistic planner ($\beta = 10$). The top row shows the latent trajectory in PCA space, both for each head's predicted trajectory, the mean trajectory across heads, and the true environment. The bottom row shows the estimated return of each trajectory at each depth.

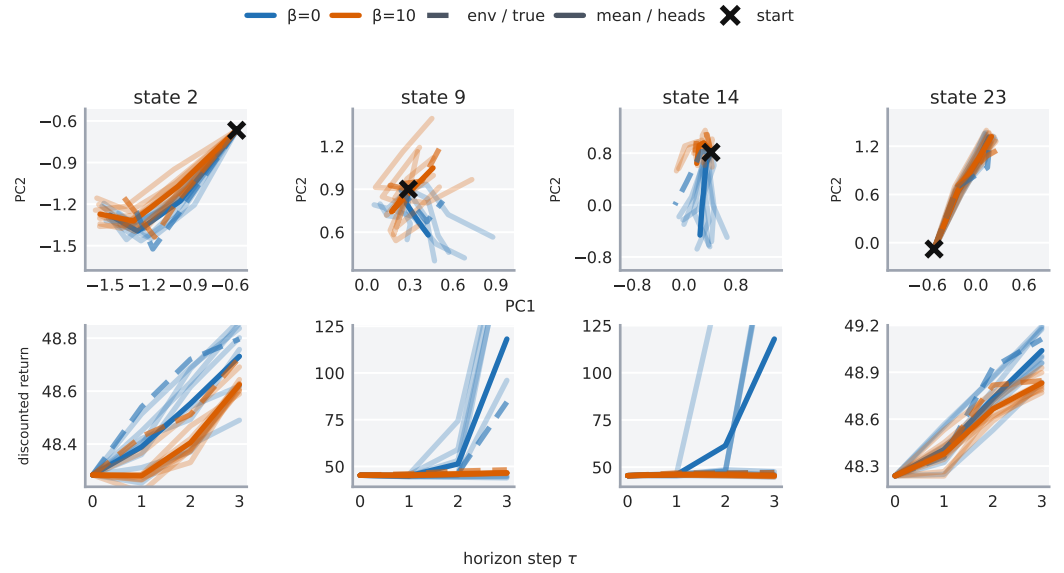


Figure A.9: Latent trajectory gallery on quadruped-walk at 50k (four states). Same as Figure A.8 but for quadruped-walk. Note that for some states the estimated value increases significantly.

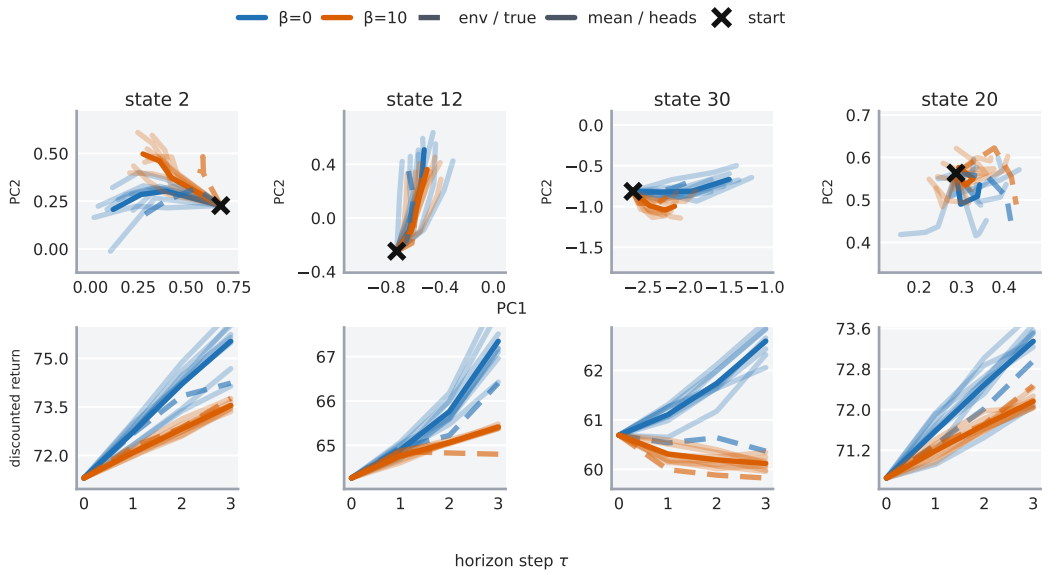


Figure A.10: Latent trajectory gallery on dog-stand at 50k (four states). Same as Figure A.8 but for dog-stand.

Across all three tasks, the state-level gallery makes the uncertainty effect concrete. Even when the predicted return stays competitive, the pessimistic planner tends to keep the per-head rollout bundle tighter than the optimistic planner, which matches the reduction in value disagreement highlighted in the main paper.

A.3.3. Task visualization

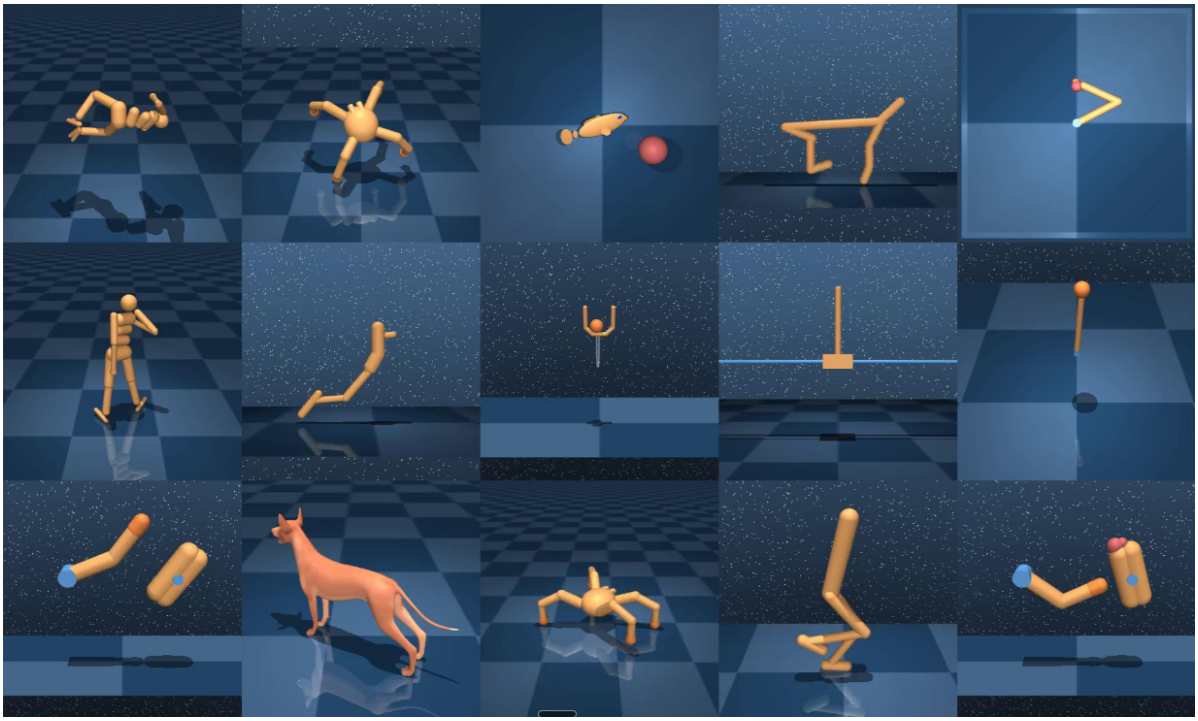


Figure A.11: DMControl tasks visualization. Images of all the embodiments we control in the DMControl tasks. The tasks include controlling them to run, walk, jump, balance, reach, and perform actions like swing-up and spin, covering a diverse range of continuous control scenarios.

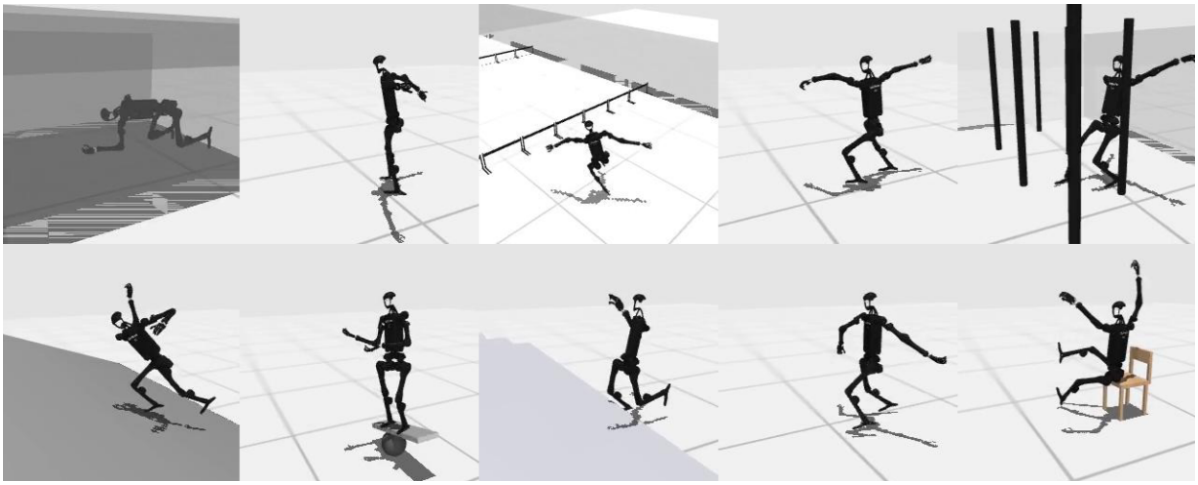


Figure A.12: HumanoidBench locomotion suite visualization. Images of the Unitree robot we control in the HumanoidBench locomotion suite. The tasks include running, walking, crawling, balancing, sitting, reaching, and performing actions like walking on stairs or walking while avoiding collisions with poles, which cover a diverse range of robotic locomotion scenarios.

B

Training a Value-Equivalent Dynamics Ensemble

To the best of our knowledge, EfficientTDMPC is the first method to use a dynamics ensemble in the context of value-equivalent world models. The paper uses this ensemble for trajectory sampling, which provides both return estimates and a disagreement-based uncertainty signal. This appendix briefly describes how the ensemble is trained and gives an informal interpretation of why it may improve value-equivalent prediction. The ensemble is trained with the same objectives as BMPC, but with multiple dynamics heads instead of a single dynamics model. Each dynamics head predicts the next latent state for the same replay-buffer transition. The reward and value losses are then evaluated on rollouts produced by the different dynamics heads, and the resulting losses are averaged before updating the shared components and prediction heads. This training procedure may improve robustness because the reward and value heads must learn to produce accurate predictions under several types of dynamics error. In a single-head model, the value-equivalent objective can adapt to the specific errors made by one learned dynamics model. With an ensemble, the downstream reward and value heads are exposed to multiple plausible latent transitions. This may make the learned return estimates less sensitive to the error pattern of any one dynamics head. The ensemble may also encourage the shared encoder to learn more stable latent representations. Since several dynamics heads must operate on the same encoded states, the latent space is pressured to support multiple compatible transition models rather than only one. This interpretation is informal, but it provides one possible explanation for why a dynamics ensemble can be useful even when the world model is trained for value equivalence rather than reconstruction.

C

Additional Experimental Results

This appendix reports the numeric values underlying the additional result tables. AUC tables report normalized area under the learning curve, and final-performance tables report the final return at the benchmark budget. Task-level final-performance rows use raw final return, whereas aggregate final-performance rows use normalized aggregate final return. Missing entries indicate method-task combinations that are not available in the generated CSV. Aggregate entries are reported only when all tasks needed for that aggregate are available for the method; otherwise, the aggregate is marked as unavailable. Higher values indicate better sample efficiency or stronger final performance. The best-performing method in each row is shown in **bold**, and the second-best method is shown in *italics*.

C.1. Comparison Against Baselines

Table C.1: Easy DMC AUC comparison at 200k environment steps. At least three seeds were used per task. Task rows report normalized AUC; the final row reports normalized AUC over the benchmark tasks.

Task or aggregate	Ours	BMPC	TD-MPC2	DreamerV3	SAC	SimBa-v2	MRS.Q	BOOM	BRC	EZ-V2
cup-catch	0.149	0.150	0.150	0.133	0.149	**0.156**	*0.151*	–	–	0.147
pendulum-swingup	**0.150**	0.145	0.135	0.127	0.027	0.148	0.064	–	–	*0.149*
cartpole-swingup-sparse	**0.331**	0.123	0.042	0.168	0.007	*0.255*	0.086	–	–	0.167
finger-spin	0.105	**0.146**	*0.146*	0.060	0.119	0.116	0.098	–	–	0.111
hopper-hop	**0.154**	0.075	0.041	0.015	0.000	*0.123*	0.044	–	–	0.072
acrobot-swingup	**0.217**	0.135	*0.136*	0.079	0.005	0.083	0.042	–	–	0.104
cartpole-swingup	0.150	0.144	0.146	0.130	0.144	*0.152*	**0.153**	–	–	0.135
hopper-stand	**0.158**	0.075	0.018	0.035	0.003	0.101	0.053	–	–	*0.125*
reacher-hard	0.130	0.126	0.100	*0.145*	0.030	**0.158**	0.089	–	–	0.128
cartpole-balance	0.151	0.160	0.163	0.154	**0.172**	*0.170*	0.160	–	–	0.138
cartpole-balance-sparse	0.150	0.151	**0.152**	0.150	0.126	0.122	*0.152*	–	–	0.150
cheetah-run	0.142	0.133	0.136	0.085	0.098	*0.149*	**0.153**	–	–	0.138
finger-turn-easy	**0.168**	0.133	0.120	0.084	0.064	*0.145*	0.106	–	–	0.059
finger-turn-hard	**0.153**	0.087	0.122	0.052	0.019	*0.129*	0.058	–	–	0.038
fish-swim	0.080	0.113	0.159	0.079	0.058	**0.318**	*0.212*	–	–	–
quadruped-run	0.129	0.104	0.124	0.038	0.029	**0.229**	*0.172*	–	–	0.097
quadruped-walk	0.104	0.108	0.143	0.011	0.023	**0.198**	*0.159*	–	–	0.120
reacher-easy	0.143	0.135	**0.153**	0.148	0.095	*0.149*	0.129	–	–	0.139
walker-run	**0.150**	*0.144*	0.139	0.057	0.073	0.135	0.115	–	–	0.085
walker-stand	0.151	*0.156*	**0.157**	0.146	0.079	0.153	0.153	–	–	0.145
walker-walk	0.149	**0.150**	*0.150*	0.108	0.121	0.150	0.149	–	–	0.143
Normalized aggregate	*0.153*	0.128	0.125	0.095	0.069	**0.159**	0.119	–	–	–

Table C.2: Easy DMC final-performance comparison at 200k environment steps. At least three seeds were used per task. Task rows report raw final return; the final row reports normalized final return over the benchmark tasks.

Task or aggregate	Ours	BMPC	TD-MPC2	DreamerV3	SAC	SimBa-v2	MRS.Q	BOOM	BRC	EZ-V2
cup-catch	**983.7**	980.9	976.6	866.2	968.1	*983.7*	966.2	–	–	980.8
pendulum-swingup	837.0	**841.4**	824.5	733.1	267.9	*839.0*	443.1	–	–	837.6
cartpole-swingup-sparse	**842.4**	370.9	308.2	614.9	54.80	*779.3*	342.3	–	–	712.4
finger-spin	699.6	**982.4**	*974.4*	508.7	807.2	788.6	710.3	–	–	753.5
hopper-hop	**331.9**	*233.7*	166.6	51.17	0.00	221.3	117.5	–	–	152.1
acrobot-swingup	**403.5**	242.3	296.3	177.7	3.13	192.5	138.5	–	–	*307.6*
cartpole-swingup	*876.6*	875.5	871.9	795.6	842.0	**880.4**	860.5	–	–	789.7
hopper-stand	**900.5**	*872.0*	261.5	313.9	46.10	749.5	448.9	–	–	832.8
reacher-hard	*882.8*	860.5	870.9	866.6	197.2	**936.8**	627.9	–	–	808.5
cartpole-balance	993.5	992.5	995.6	948.5	*998.0*	**999.7**	997.0	–	–	933.0
cartpole-balance-sparse	**1000.0**	*1000.0*	1000.0	958.1	806.9	1000.0	1000.0	–	–	998.5
cheetah-run	600.7	611.4	614.4	353.4	573.7	623.6	**713.4**	–	–	*659.2*
finger-turn-easy	**935.1**	797.2	768.0	542.2	100.5	*831.0*	583.8	–	–	388.9
finger-turn-hard	**870.6**	752.9	797.1	377.7	124.3	*804.7*	452.9	–	–	239.7
fish-swim	248.0	268.8	378.9	183.8	57.67	**617.2**	*512.4*	–	–	–
quadruped-run	*842.1*	578.9	759.0	69.47	76.63	**880.1**	800.7	–	–	503.3
quadruped-walk	734.6	732.8	*914.7*	58.03	74.10	**951.4**	795.0	–	–	893.8
reacher-easy	886.5	*960.6*	**976.2**	844.5	714.6	920.8	851.9	–	–	900.7
walker-run	**827.2**	*809.1*	780.5	377.3	488.8	772.4	700.8	–	–	553.8
walker-stand	**986.3**	980.6	977.7	931.3	738.7	*980.9*	979.8	–	–	969.4
walker-walk	968.3	*973.2*	**973.4**	802.3	922.5	969.9	962.8	–	–	965.2
Normalized aggregate	**1.132**	1.000	1.001	0.706	0.480	*1.132*	0.909	–	–	–

Table C.3: Hard DMC AUC comparison at 600k environment steps. At least three seeds were used per task. Task rows report normalized AUC; the final row reports normalized AUC over the benchmark tasks.

Task or aggregate	Ours	BMPC	TD-MPC2	DreamerV3	SAC	SimBa-v2	MRS.Q	BOOM	BRC	EZ-V2
humanoid-run	**0.578**	0.204	0.131	0.002	0.002	0.263	0.108	*0.304*	0.183	–
humanoid-stand	**0.372**	0.263	0.144	0.004	0.003	*0.308*	0.137	0.277	0.206	–
humanoid-walk	**0.419**	0.224	0.084	0.001	0.001	*0.255*	0.128	0.162	0.140	–
dog-trot	**0.398**	0.267	0.070	0.004	0.006	0.293	*0.336*	0.124	0.176	–
dog-walk	**0.435**	0.365	0.200	0.004	0.008	*0.371*	0.363	0.337	0.294	–
dog-run	*0.381*	0.316	0.131	0.005	0.011	0.301	**0.388**	0.349	0.215	–
dog-stand	**0.495**	0.435	0.328	0.018	0.028	*0.480*	0.466	0.416	0.393	–
Normalized aggregate	**0.440**	0.296	0.156	0.005	0.008	*0.324*	0.275	0.281	0.230	–

Table C.4: Hard DMC final-performance comparison at 600k environment steps. At least three seeds were used per task. Task rows report raw final return; the final row reports normalized final return over the benchmark tasks.

Task or aggregate	Ours	BMPC	TD-MPC2	DreamerV3	SAC	SimBa-v2	MRS.Q	BOOM	BRC	EZ-V2
humanoid-run	**498.4**	245.8	120.2	0.70	1.07	193.8	137.7	*297.1*	146.6	–
humanoid-stand	656.9	*878.0*	475.8	4.10	5.83	**883.4**	592.7	839.2	683.4	–
humanoid-walk	**944.1**	*823.6*	436.9	0.70	1.77	586.9	438.1	733.4	398.2	–
dog-trot	**893.3**	*873.8*	210.7	4.70	8.13	746.1	811.0	501.7	519.0	–
dog-walk	**938.8**	926.8	644.8	6.50	11.37	882.6	880.5	*930.7*	827.3	–
dog-run	*509.7*	459.6	172.9	0.00	9.70	427.7	**510.8**	505.9	309.7	–
dog-stand	**965.0**	*956.6*	691.1	21.70	74.43	954.6	954.2	795.8	941.0	–
Normalized aggregate	**1.154**	*1.000*	0.514	0.006	0.019	0.892	0.822	0.938	0.715	–

Table C.5: HumanoidBench-Hard AUC comparison at 1M environment steps. At least three seeds were used per task. Task rows report normalized AUC; the final row reports normalized AUC over the benchmark tasks.

Task or aggregate	Ours	BMPC	TD-MPC2	DreamerV3	SAC	SimBa-v2	MRS.Q	BOOM	BRC	EZ-V2	PPO
h1hand-walk	**0.965**	0.471	0.136	0.020	0.011	0.126	0.534	*0.679*	0.239	–	0.006
h1hand-stand	**1.037**	0.718	0.117	0.039	0.034	0.230	*0.894*	0.749	0.389	–	0.005
h1hand-run	**4.505**	0.707	0.162	0.045	0.047	0.276	*2.443*	1.327	0.613	–	0.038
h1hand-crawl	*0.777*	0.704	0.632	0.256	0.351	**0.795**	0.687	–	0.463	–	0.005
h1hand-hurdle	**0.694**	*0.523*	0.121	0.027	0.033	0.136	0.397	0.491	0.299	–	0.024
h1hand-maze	**1.009**	*0.886*	0.378	0.356	0.408	0.417	0.733	–	0.702	–	0.016
h1hand-sit-simple	**1.119**	0.734	0.511	0.035	0.129	0.958	*1.062*	0.881	0.740	–	0.006
h1hand-sit-hard	0.752	0.561	0.096	0.016	0.044	*0.857*	**1.006**	–	0.307	–	0.007
h1hand-slide	**0.928**	0.577	0.183	0.039	0.031	0.304	0.581	*0.667*	0.268	–	0.012
h1hand-stair	*0.346*	**0.451**	0.122	0.047	0.040	0.327	0.334	–	0.127	–	0.014
h1hand-pole	**1.078**	0.731	0.147	0.055	0.078	0.445	*0.935*	0.493	0.321	–	0.007
h1hand-balance-simple	**1.368**	0.445	0.063	0.039	0.090	0.480	*1.095*	–	0.254	–	0.016
h1hand-reach	0.323	*0.545*	0.516	0.124	0.203	**0.729**	0.399	–	0.463	–	0.001
Normalized aggregate	**1.146**	0.619	0.245	0.084	0.115	0.468	*0.854*	–	0.399	–	0.012

Table C.6: HumanoidBench-Hard final-performance comparison at 1M environment steps. At least three seeds were used per task. Task rows report raw final return; the final row reports normalized final return over the benchmark tasks.

Task or aggregate	Ours	BMPC	TD-MPC2	DreamerV3	SAC	SimBa-v2	MRS.Q	BOOM	BRC	EZ-V2	PPO
h1hand-walk	**921.3**	757.0	234.1	19.49	12.54	357.8	812.1	*900.0*	442.1	–	5.36
h1hand-stand	**938.9**	805.8	192.7	40.94	38.08	638.4	*923.4*	908.7	621.5	–	5.36
h1hand-run	**786.7**	116.0	28.51	6.39	7.38	112.4	*784.0*	330.3	148.2	–	5.36
h1hand-crawl	*927.8*	921.2	896.8	255.8	459.2	**955.6**	860.6	–	533.2	–	5.36
h1hand-hurdle	**220.3**	179.4	38.89	6.22	7.84	92.95	128.5	*184.8*	91.57	–	5.36
h1hand-maze	**325.2**	279.6	133.4	112.8	139.6	147.4	294.9	–	*312.4*	–	5.36
h1hand-sit-simple	**921.7**	722.3	607.1	47.87	201.1	913.5	*920.7*	896.6	785.5	–	5.36
h1hand-sit-hard	605.8	600.2	139.2	15.37	58.16	*718.1*	**861.0**	–	489.2	–	5.36
h1hand-slide	**580.1**	357.4	78.83	20.63	18.23	282.4	337.3	*471.3*	186.7	–	5.36
h1hand-stair	192.6	*316.2*	43.11	16.19	17.39	**339.3**	188.5	–	51.21	–	5.36
h1hand-pole	*805.2*	605.8	98.70	47.81	86.89	561.3	**833.9**	688.4	410.3	–	5.36
h1hand-balance-simple	**449.6**	271.3	24.94	12.62	38.98	178.0	*407.0*	–	85.72	–	5.36
h1hand-reach	1978.4	*4051.6*	3610.5	863.9	1391.8	**5320.5**	2989.6	–	2870.9	–	5.36
Normalized aggregate	**1.581**	1.000	0.387	0.107	0.175	0.887	*1.504*	–	0.702	–	0.015

D

Later Contributions

This appendix contains two additional results that were obtained after the EfficientTDMPC was submitted to NeurIPS. They will be integrated into the camera-ready version of the EfficientTDMPC paper.

D.1. Improved Component Ablations

This appendix presents newer component ablation results. The main difference is that the AUC is plotted rather than the learning curves. Additionally, we found that the confidence interval was underestimated in the original component ablation plots. The new component ablation results are shown in Fig. D.1.

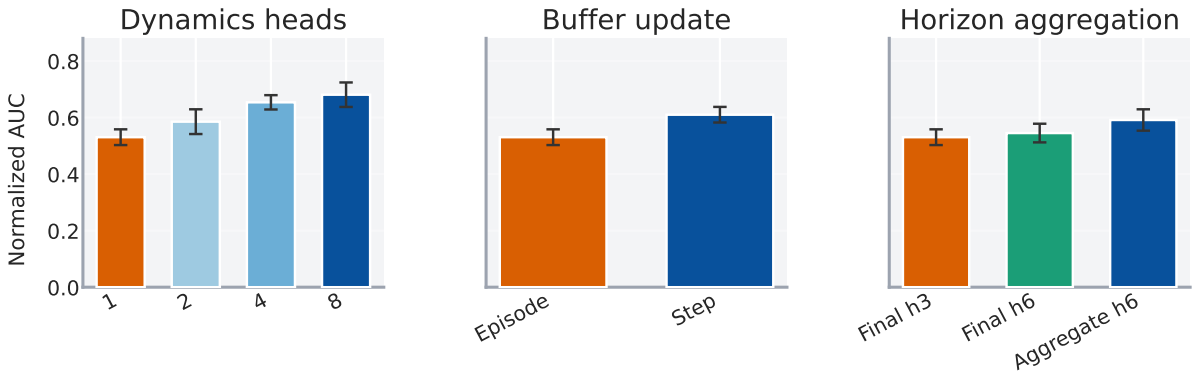


Figure D.1: Direct normalized-AUC summary of the component ablations. For each task and seed, we compute the area under the normalized learning curve over budget fraction, then average those seed-level AUC values within each task and propagate the corresponding task-level standard errors through the mean over tasks. Bars show aggregate mean normalized AUC and error bars show propagated 95% confidence intervals.

These results change the conclusion of the EfficientTDMPC paper to some extent. The plot indicates that the individual components did make statistically significant improvements by themselves already. The dynamics ensemble especially yields significant gains, which consistently increase with the number of ensemble members. The buffer update interval and multi-depth returns also yield weakly significant improvements. These new results do not change the rest of the conclusions, but they suggest that

the final performance gains are less due to component interactions and more due to the individual contributions themselves.

D.2. Novelty of Dynamics Ensembles in MuZero-style Value Equivalent World Models

After additional literature review, to our knowledge, EfficientTDMPC is actually not just the first method to introduce dynamics ensembles to the TD-MPC family, but in fact the first method to introduce dynamics ensembles to any value-equivalent world model method. Together with the ablation above, we therefore are the first to show that dynamics ensembles can improve the performance of value-equivalent world models significantly.

E

Software

The code for EfficientTDMPC and the experiments will be made available at `https://github.com/Thomasevers2517/EfficientTDMPC`. The code will be released after the EfficientTDMPC paper is accepted at NeurIPS.

The codebase also contains the source code for a tool developed during this thesis. In order to save compute, this tool takes training and replay buffer checkpoints, and is able to run the planner on the spot. The tool is able to change all the hyperparameters and other details of the planner, and inspect all the steps of the planning process. Furthermore, it is very easy to add new planner functionality and test it on the existing checkpoints. The tool can be used in future work to test and understand new ideas without requiring the compute infrastructure to run full training seeds with different planner implementations.