Fac-TDMPC: Learning an Efficient and Robust Factored World Model for Robot Planning

Yuan Zhang
University of Freiburg, Germany
yzhang@cs.uni-freiburg.de

Jianhong Wang
University of Bristol, UK
jianhong.wang@bristol.ac.uk

Joschka Boedecker
University of Freiburg, Germany
jboedeck@cs.uni-freiburg.de

Abstract: Model-based reinforcement learning (MBRL) achieves high sample efficiency in robotics, but existing methods such as TDMPC suffer from high planning latency due to the combination of a centralized world model and model predictive control as planning methods, thus limiting real-time deployment in high-dimensional action spaces. We propose Fac-TDMPC, which learns a factored latent-space world model and thus enables decentralized MPC, which largely reduces the optimization complexity in theory. Experiments on robotic control tasks show that Fac-TDMPC achieves substantial planning speedups, while maintaining control performance and improving robustness to action perturbations.

Keywords: Factored World Model, Decentralized Planning, Model-based Reinforcement Learning

1 Introduction

Reinforcement learning (RL) [1] is a powerful paradigm for solving sequential decision-making problems, achieving remarkable success across diverse domains, such as mastering the game of Go [2] and controlling quadruped robots [3]. Model-based reinforcement learning (MBRL) [4] is a prominent branch of RL that offers superior sample efficiency and improved interpretability compared to model-free approaches [5, 6]. TDMPC [7] and its successor TDMPC2 [8] represent state-of-the-art MBRL methods in robotics. These frameworks learn a world model for representing robotic systems in latent spaces and perform trajectory optimization using model predictive control (MPC) in the latent space, enabling long-horizon planning and constraint handling.

Despite their successes, TDMPC-based frameworks typically employ a centralized world model with densely connected neural networks as dynamics, reward and value functions, resulting in an MPC optimization space of $\mathcal{O}(|\mathcal{A}|^H)$, where $\mathcal{A}$ denotes the action space and H is the planning horizon. For high-dimensional action spaces, this leads to substantial computational overhead during planning, often referred as *the curse of dimensionality* [9], which restricts their scalability to more complex robotic scenarios and real-time deployment.

To overcome the planning latency during deployment, we propose **Fac-TDMPC**, a novel extension of TDMPC that incorporates **a factored world model** in the latent space. Instead of the densely connected neural networks used in TDMPC, we employ factored dynamics, reward, and value functions for each action dimension i, following the factored decentralized Markov decision processes (fDec-MDPs) framework [10]. This latent world model enables decentralized MPC execution, reducing the optimization space to $\mathcal{O}(\sum_{i=1}^{N} |\mathcal{A}^i|^H)$, where $\mathcal{A}^i$ denotes the action space of dimension i, thereby substantially lowering optimization complexity. To learn the factored world model, we employ a model distillation technique [11] to transfer reward and value predictions from a central-

9th Conference on Robot Learning (CoRL 2025), Seoul, Korea.

ized expert model. An ideal target in distilling models for planning is for the student model to preserve the expert model's prediction ordering over actions. We achieve this by perturbing optimal action sequences with Gaussian noise and minimize the KL divergence between the distributions predicted by their respective reward and value functions.

Our contributions are as follows: (1) Fac-TDMPC — a novel MBRL framework that learns a factored world model to enable efficient planning in robotic tasks; (2) A model distillation procedure that transfers knowledge from a centralized expert world model (e.g., TDMPC) to a factored student world model while preserving the prediction ordering over actions; (3) Empirical evidence showing that the proposed architecture achieves significant planning speedups, while maintaining the control performance and improving robustness to action perturbations.

2 Related Work

Model-based reinforcement learning (MBRL) [4] integrates planning and learning by building a predictive model of the environment's dynamics and using it for decision-making, often achieving superior sample efficiency compared to model-free methods [5, 6, 7, 12]. Most existing approaches learn either reconstruction-based models [12] or latent-space models with state abstractions [7, 8], which are used for planning at deployment. However, when unstructured latent-space models are combined with computationally heavy planners such as model predictive control (MPC), inference becomes slow, limiting real-time applicability in robotics. Our work, Fac-TDMPC, addresses this by learning factored world models and using decentralized planning to accelerate decision-making.

Our method requires learning a factored world model on which the latent state and action spaces are factored. The factorization idea has been explored in model-based planning settings. Some works [13, 14, 15] assumed full knowledge of the system, which is limited to certain environmental domains. Jiang et al. [16] tried to learn models with action abstractions, but requiring a variational autoencoder to reconstruct the original actions. In contrast, our method learns the factored world on latent space, without the prior knowledge of the system, and speeds up planning without action reconstruction.

Our paper utilizes a decentralized MPC method [17] to efficiently plan on the factored world model. More efficient planning methods adapted on different special structures exist. Variable elimination [18] and max-plus [19] are exact and approximated decentralized planning methods on coordination graph [18]. Liu et al. [20] adopted mixed-integer programming for sparsified neural networks. Our method can be extended with these advanced planning methods by incorporating a special structure in the latent space.

Our approach is also related to multi-agent reinforcement learning (MARL) [9], where high-dimensional action spaces present similar challenges. In robotics, some methods decompose a single robot into multiple agents and apply MARL for control [21, 22]. However, these approaches require specifying factored state and action spaces *a priori*, which can be restrictive and suboptimal. We instead learn the factored latent world model automatically, serving as an abstraction for efficient planning. In this work, we adopt reward and value decomposition from VDN [23] as a proof of concept, though more advanced MARL decompositions, such as QMIX [24] or QTRAN [25], could be integrated in future work.

3 Preliminaries

3.1 Factored Dec-MDP

A Markov decision process (MDP) [26] is defined by the tuple $\langle \mathcal{S}, \mathcal{A}, T, R, \mu_0, \gamma \rangle$, where $\mathcal{S}$ and $\mathcal{A}$ denote the state and action spaces, $\Delta_{\mathcal{S}}$ and $\Delta_{\mathcal{A}}$ are the space of probability measure, $T : \mathcal{S} \times \mathcal{A} \to \Delta_{\mathcal{S}}$ is the transition function, $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, $\mu_0 \in \Delta_{\mathcal{S}}$ is the initial state distribution, and $\gamma \in [0, 1]$ is the discount factor. A factored decentralized MDP (fDec-MDP) [10] is a special case of the MDP to model multi-agent systems, where $\mathcal{S} = \times_{i=1}^{N} \mathcal{S}^i$ and $\mathcal{A} = \times_{i=1}^{N} \mathcal{A}^i$

partition state and action spaces into N disjoint subspaces. The fDec-MDP is said to be transition independent if $T(s_{h+1}|s_h, a_h) = \prod_{i=1}^{N} T_i(s^i_{h+1}|s^i_h, a^i_h)$, and reward independent if $R(s_h, a_h) = f(R_1(s^1_h, a^1_h) \dots R_N(s^N_h, a^N_h))$, where f is a monotonic function, $s^i_h, s^i_{h+1} \in \mathcal{S}^i, a^i_h \in \mathcal{A}^i$.

3.2 TDMPC

TDMPC [7] is a model-based RL algorithm that learns a world model in a latent space which is aimed to facilitate real-time planning. Given that the agent receives observations o in the observation space $\mathcal{O}$, it internally maintains an MDP in the latent space $\mathcal{S}$, which consists of the following components: (1) an encoder $s = f_E(o)$ maps observations to latent states; (2) a transition function $s' = f_T(s, a)$ models the dynamics in the latent space; (3) a reward function $\hat{r} = f_R(s, a)$ predicts the reward on the latent space; (4) a terminal value function $\hat{q} = f_Q(s, a)$ predicts the discounted sum of rewards, where $s, s' \in \mathcal{S}, a \in \mathcal{A}, o \in \mathcal{O}$.

TDMPC maintains a replay buffer $\mathcal{B}$ with collecting trajectories of horizon H through interaction with the environment, and iteratively updates the world model using the data sampled from $\mathcal{B}$. The f_E, f_T, f_R, f_Q components are jointly optimized to minimize the following objective function:

$$\mathcal{L}(f_E, f_T, f_R, f_Q) = \mathbb{E}_{(o,a,r)_{0:H+1} \sim \mathcal{B}} \Big[\sum_{h=0}^{H} \lambda^h \Big(\|s_{h+1} - \text{sg}(f_E(o_{h+1}))\|_2^2 + \|\hat{r}_h - r_h\|_2^2 + \|\hat{q}_h - q_h\|_2^2 \Big) \Big], \tag{1}$$

where $s_0 = f_E(o_0), s_{h+1} = f_T(s_h, a_h), \hat{r}_h = f_R(s_h, a_h), \hat{q}_h = f_Q(s_h, a_h)$ are predictions of the model. sg is the stop-gradient operation, $q_h = r_h + \max_a \gamma f_{\bar{Q}}(f_E(o_{t+1}), a)$ is the TD-target ($f_{\bar{Q}}$ is an exponential moving-average version of f_Q), $\lambda \in [0, 1]$ is a hyperparameter to weigh less on further-step learning. TDMPC utilizes the learned world model by planning on the latent state space with Model Predictive Path Integral (MPPI) [27]. Specifically, at each decision step t, the observation o_t is first transformed into the corresponding latent state as $s_t = f_E(o_t)$, and MPPI is leveraged to efficiently find the maximized solution of H-step return $G^H(s_t, a_{t:t+H})$ as follows:

$$\begin{aligned} a^*_{t:t+H} = \underset{a_{t:t+H}}{\text{argmax}} \quad & G^H(o_t, a_{t:t+H}) = \sum_{h=t}^{t+H-1} \gamma^{h-t} f_R(s_h, a_h) + \gamma^H f_Q(s_{t+H}, a_{t+H}), \\ & \text{s.t.} \quad s_t = f_E(o_t), s_{h+1} = f_T(s_h, a_h). \end{aligned} \tag{2}$$

The optimal solution of step t's action a^*_t is acquired and executed in the environment.

4 Method

TDMPC [7] learns a world model in the latent space on which an MDP is defined. In this paper, we instead learn a factored world model in the latent space that admits a fDec-MDP. This factorization can thus enable efficient planning at inference time, overall denoted as the Fac-TDMPC method.

4.1 fDec-MDP Model

We first assume each action dimension of the robot as an independent agent, yielding $N = \dim(\mathcal{A})$ agents. We aim to represent this N-agent system as a fDec-MDP model (introduced in Section 3.1) in the TDMPC framework. To do so, we design the factored latent space as $\mathcal{S} = \times_{i=1}^{N} \mathcal{S}_i : \mathbb{R}^{mN}$, where m is the dimension for a single agent's hidden state. The encoder module is the same as the one used in TDMPC as $s_t = f_E(o_t)$.

In order to fully decompose agents for effiicient planning, we further assume that the fDec-MDP is transition-independent so that for each agent i, it maintains an independent local transition function $s^i_{t+1} = f_{T_i}(s^i_t, a^i_t)$. Meanwhile, each agent maintains an independent reward function and a value function, written as $r^i_t = f_{R_i}(s^i_t, a^i_t)$ and $q^i_t = f_{Q_i}(s^i_t, a^i_t)$. The global reward describing the system-wise performance is the sum of all individual rewards: $r_t = f_R(s_t, a_t) = \sum_{i=1}^{N} f_{R_i}(s^i_t, a^i_t)$. As mentioned by Oliehoek and Amato [10], transition independence and reward decomposition ensure the value function is naturally decomposable, written as $q_t = f_Q(s_t, a_t) = \sum_{i=1}^{N} f_{Q_i}(s^i_t, a^i_t)$.

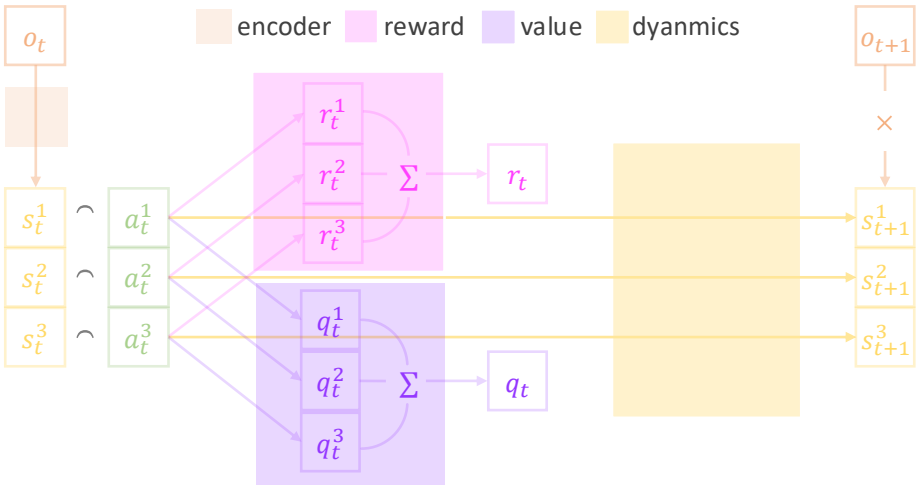


Figure 1: fDec-MDP modelling in the TDMPC framework. The observation o_t is first encoded to the latent state s_t by the encoder function $s_t = f_E(o_t)$. The latent state is factored to $N = \dim(\mathcal{A})$ agents as $s_t = \times_{i=1}^N s_t^i$. On each action dimension i, the independent dynamics function f_{T_i}, reward function f_{R_i}, value function f_{Q_i} are defined locally. The global reward and value can be summed up by the local ones.

In summary, the complete factored world model consists of a centralized encoder function f_E, a factored dynamics function $f_T = (\times_{i=1}^N f_{T_i})$, a factored reward function $f_R = (\times_{i=1}^N f_{R_i})$, and the factored value function $f_Q = (\times_{i=1}^N f_{Q_i})$, as illustrated in Figure 1.

4.2 Decentralized Optimization

As one of the main motivations for adopting the factored model, we can further achieve an efficient model predictive control method on the fly. Following the H-step return in TDMPC, we can similarly define an individual H-step return for agent i with its individual dynamics, reward and value functions as $G_i^H(o_t, a_{t:t+H}^i) = \sum_{h=t}^{t+H-1} \gamma^{h-t} f_{R_i}(s_h^i, a_h^i) + \gamma^H f_{Q_i}(s_{t+H}^i, a_{t+H}^i)$, s.t.$s_t^i = f_E(o_t)^i, s_{h+1}^i = f_{T_i}(s_h^i, a_h^i)$, which is underpinned by the following proposition:

Proposition 1 (Individual Global Max) *For the fDec-MDP system with independent transition, reward and value functions, the H-step return $[G_i^H]$ satisfies the individual global max [25] under observation o_t, which equivalently says:*

$$a_{t:t+H}^* = \underset{a_{t:t+H}}{argmax} \quad G^H(o_t, a_{t:t+H}) = \times_{i=1}^N \underset{a_{t:t+H}^i}{argmax} \quad G_i^H(o_t, a_{t:t+H}^i). \tag{3}$$

The proposition results from the additivity of the reward and value functions and the independence of the dynamics function. It implies that each action dimension can be optimized in a decentralized manner, without impairing the system-wise performance. For such factored model, the optimization space of an MPC has been reduced from $\mathcal{O}(|\mathcal{A}|^H) = \mathcal{O}(\times_{i=1}^N |\mathcal{A}^i|^H)$ to $\mathcal{O}(\sum_{i=1}^N |\mathcal{A}^i|^H)$. Besides, if given enough computational resources, the planning process of each action dimension can be executed in parallel and thus speed up N times.

4.3 Model Distillation

Since our goal is to find a more efficient factored model (f_E, f_T, f_R, f_Q) to replace the original expert system, it is reasonable to assume that the knowledge of the original expert model $(f_{E_e}, f_{T_e}, f_{R_e}, f_{Q_e})$ is known. The specific procedure is illustrated in Algorithm 1.

Algorithm 1 Model Distillation

1: **Input**: expert model $(f_{E_e}, f_{T_e}, f_{R_e}, f_{Q_e})$, dataset $\mathcal{B}$, hyperparameter λ, H
2: **Output**: factored model $(f_E, \times_{i=1}^{N} f_{T_i}, \times_{i=1}^{N} f_{R_i}, \times_{i=1}^{N} f_{Q_i})$
3: **for** iteration $i = 1, 2, ..., I$ **do**
4: Sample a H-step sequential samples $(o, a)_{0:H}$ from the dataset $\mathcal{B}$
5: Update parameters of the factored model based on the loss function as follows:
6: $\mathcal{L}(f_E, f_T, f_R, f_Q) = \mathbb{E}_{(o,a)_{0:H} \sim \mathcal{B}} \Big[\sum_{h=0}^{H} \lambda^h \Big(D_{\text{KL}}(\mathcal{R}_h, \mathcal{R}_h^e) + D_{\text{KL}}(\mathcal{Q}_h, \mathcal{Q}_h^e) \Big) \Big]$

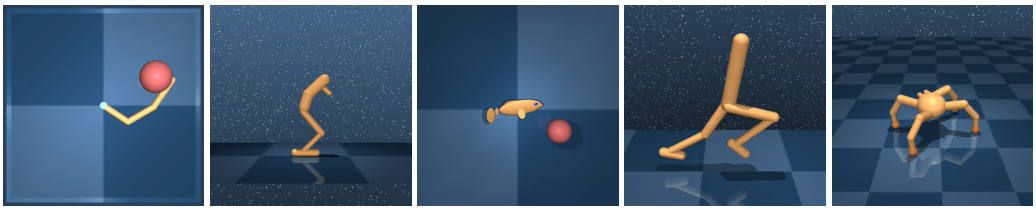
The algorithm follows standard model distillation [11]. The loss in Line 6 uses KL divergence between reward and value *distributions* predicted by the factored model and the expert model. Rusu et al. [28] observed that preserving prediction ordering over actions (which is to say $f_Q(s, a_1) > f_Q(s, a_2) \iff f_{Q_e}(s, a_1) > f_{Q_e}(s, a_2), \forall s, a_1, a_2$) can be more effective than matching raw scalar values when compressing reward/value predictors. Inspired by this evidence, for each trajectory $(o, a)_{0:H}$ we inject Gaussian noise $\epsilon_{0:H} \sim \mathcal{N}^{H+1}(0, \Sigma)$ to actions $a_{0:H}$ and propagate uncertainty through dynamics via $s_{h+1} = f_T(s_h, a_h + \epsilon_h)$. We then define soft action distributions based on reward and value predictions:

$$\mathcal{R}_h(a_h + \epsilon_h) \propto \exp\big(f_R(s_h, a_h + \epsilon_h)/T\big), \quad \mathcal{Q}_h(a_h + \epsilon_h) \propto \exp\big(f_Q(s_h, a_h + \epsilon_h)/T\big), \tag{4}$$

where T is a temperature controlling the softness of the target [11]. The expert distributions $\mathcal{R}_h^e$ and $\mathcal{Q}_h^e$ are defined analogously. We minimize the KL divergence between these distributions to preserve the prediction ordering over actions in the expert model. In practice, we use 10 noise samples to approximate the KL terms. Compared with Equation 1, we do not apply an explicit consistency loss since we target a state abstraction which is effective for planning, but not equivariant for self-predictions [29]. This difference is also noted in Figure 1.

5 Experiments

5.1 Experimental Setup



(a) Reacher Three $\dim(\mathcal{O}) = 8$, $\dim(\mathcal{A}) = 3$ (b) Hopper $\dim(\mathcal{O}) = 15$, $\dim(\mathcal{A}) = 4$ (c) Fish $\dim(\mathcal{O}) = 24$, $\dim(\mathcal{A}) = 5$ (d) Walker $\dim(\mathcal{O}) = 24$, $\dim(\mathcal{A}) = 6$ (e) Quadruped $\dim(\mathcal{O}) = 78$, $\dim(\mathcal{A}) = 12$

Figure 2: Visualization of robots used in the experiments, with increased complexity.

Task Setup. We evaluate our method in MuJoCo [30] simulator. We select five robots with increasing state and action complexity as shown in Figure 2. The selection follows TDMPC [7], but we only keep robots with more than three action dimensions to validate the factored design; this yields eight tasks (see Appendix A).

Expert and Dataset Collection. We train TDMPC with default hyperparameters [7] on each task. The final checkpoint serves as the expert model, and its replay buffer of 10^6 transitions is used as the offline dataset.

Training hyperparameters. We perform model distillation (Sec. 4.3) for 10^5 optimization steps and evaluate the factored model every 5×10^3 steps over 10 episodes. The TDMPC latent dimension is 512 in all experiments; the factored model uses a per-agent latent size of $\lfloor 512/N \rfloor$ to keep parameter counts comparable. Other hyperparameters are listed in Appendix A.2.

5.2 Training Performances

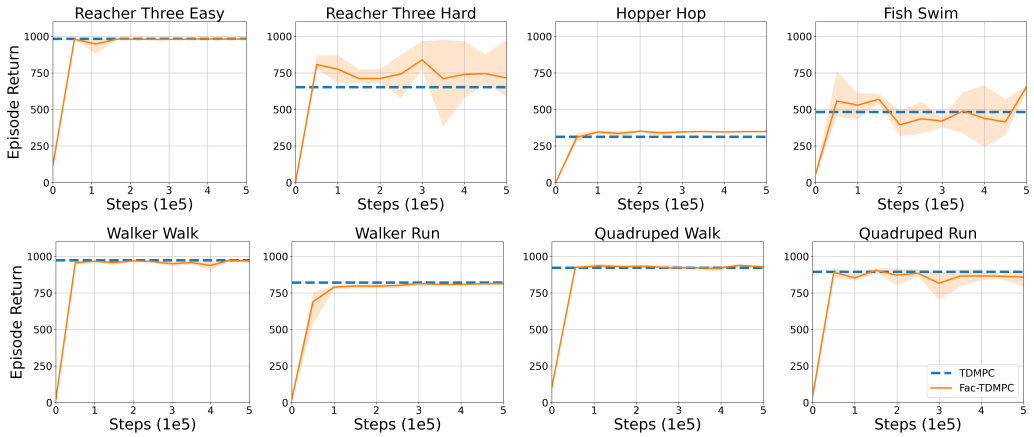


Figure 3: Learning curves of Fac-TDMPC on MuJoCo tasks. Each curve shows the average episode return over three seeds, with shaded regions denoting the standard deviation.

Figure 3 presents the learning curves of Fac-TDMPC. Nearly for all tasks, Fac-TDMPC matches or exceeds the asymptotic performance of the baseline TDMPC, demonstrating that the proposed factored architecture does not sacrifice control quality despite its reduced computational footprint (shown in the next section). In more challenging tasks, such as Quadruped Run, Fac-TDMPC converges to slightly lower final performance compared to TDMPC. We attribute this to the increased requirements on action coordination on highly dynamic tasks.

5.3 Planning Efficiency

In this section, we mainly validate the prior motivation of the proposed Fac-TDMPC, that the learned factored structure is efficient for further planning, specifically, MPPI [27] algorithm used here. For MPPI, some key choices are fundamental for the efficiency of planning: the inference speed of all models, the number of samples and iterations in optimization.

Table 1 compares the prediction time cost of TDMPC and Fac-TDMPC across 5 different robots, since the model structure only relates to the number of action dimensions instead of a specific task. We only compare the speed of (f_T, f_R, f_Q) without the encoder f_E since all encoders apply the same size of feedforward network. TDMPC is tested on a 1-thread CPU and Fac-TDMPC on N-thread CPU, depending on the robot's action dimensions N, to enable parallel prediction. Notably, the time cost for TDMPC is similar across different robots, since the same model structure is used. Across all robots and model components, Fac-TDMPC achieves a substantial reduction in per-step computation time compared to TDMPC, and the efficiency gain continues to improve with the increase in action dimensions. This proves that the speedup stems from the parallel prediction of the factored structure across multiple threads, and the size of each action dimension's individual model is much smaller than the one in TDMPC.

Table 1: Prediction time cost of different models on 5 different robots. All wall times are reported in milliseconds with average and standard deviation over 100 runs.

Methods		Reacher Three	Hopper	Fish	Walker	Quadruped
TDMPC	f_T	5.83 ± 0.01	5.86 ± 0.01	5.83 ± 0.01	5.86 ± 0.01	5.96 ± 0.01
	f_R	5.32 ± 0.01	5.34 ± 0.01	5.26 ± 0.01	5.28 ± 0.01	5.34 ± 0.01
	f_Q	11.70 ± 0.04	11.70 ± 0.04	11.70 ± 0.04	11.71 ± 0.04	11.86 ± 0.05
Fac-TDMPC	f_T	1.25 ± 0.01	0.83 ± 0.02	0.77 ± 0.01	0.721 ± 0.01	0.63 ± 0.01
	f_R	1.02 ± 0.00	0.65 ± 0.01	0.57 ± 0.01	0.48 ± 0.01	0.49 ± 0.00
	f_Q	2.56 ± 0.03	1.71 ± 0.03	1.53 ± 0.03	1.32 ± 0.03	0.96 ± 0.04

Prediction time cost determines one call of the models, while the number of calls also matters. In MPPI, more samples and iterations mean more calls to the models. We test the control performance of the Walker Run task with various numbers of samples and iterations and report in Figure 4. The default number of samples and iterations is 512 and 6. Reducing them will deteriorize the control performance of both TDMPC and Fac-TDMPC. However, the Fac-TDMPC achieves a much smaller reduction, and even achieves acceptable performance for 128 samples or 2 iterations, which could reduce 4 or 3 times computations. We attribute this to the individual global max property of the return (Proposition 1), which reduces the optimization space from $\mathcal{O}(\times_{i=1}^{N}|\mathcal{A}^i|^H)$ to $\mathcal{O}(\sum_{i=1}^{N}|\mathcal{A}^i|^H)$. This leads the factored architecture to better utilization of limited optimization resources, which is common in robotic tasks [31].

5.4 Robustness

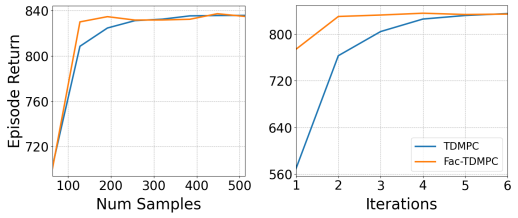


Figure 4: Control performance on Walker Run with various optimization parameters.

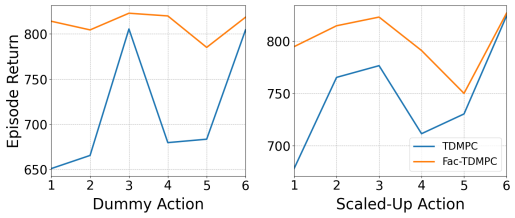


Figure 5: Control performance on Walker Run under various perturbations.

The factored structure permits independent optimization of each action dimension, which can improve robustness to action perturbations. We evaluate two perturbations: (1) one action dimension is *dummy* and follows a Normal distribution; (2) one action dimension is consistently scaled, simulating modified motor strength. Results in Figure 5 show that certain action dimensions are more sensitive to the perturbations, and Fac-TDMPC maintains more stable and higher episode returns across perturbed dimensions.

6 Conclusion

This paper introduces Fac-TDMPC, a factored world model in latent space, enabling fast decision-making on deployment and robustness for high-dimensional action scenarios. Fac-TDMPC achieves significant speedups in planning through decentralized optimization while maintaining high control performance on various robotic tasks in MuJoCo simulators. Alongside, the learned factored structure enhances robustness to action perturbations.

In future work, we aim to enhance performance on highly dynamic robotic tasks, which demand stronger coordination across multiple action dimensions. Besides, for non-factored scenarios, it is necessary to learn how to decompose the action space automatically without strictly assuming each dimension is independent.

References

[1] R. S. Sutton and A. G. Barto. *Reinforcement Learning - an Introduction*. Adaptive Computation and Machine Learning. MIT Press, 1998. ISBN 978-0-262-19398-6.

[2] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, Oct. 2017. ISSN 1476-4687. doi:10.1038/nature24270.

[3] I. Kostrikov, L. M. Smith, and S. Levine. Demonstrating a walk in the park: Learning to walk in 20 minutes with model-free reinforcement learning. In K. E. Bekris, K. Hauser, S. L. Herbert, and J. Yu, editors, *Robotics: Science and Systems XIX, Daegu, Republic of Korea, July 10-14, 2023*, 2023. doi:10.15607/RSS.2023.XIX.056.

[4] T. M. Moerland, J. Broekens, A. Plaat, and C. M. Jonker. Model-based reinforcement learning: A survey. *Foundations and Trends® in Machine Learning*, 16(1):1–118, Jan. 2023. ISSN 1935-8237, 1935-8245. doi:10.1561/2200000086.

[5] W. Ye, S. Liu, T. Kurutach, P. Abbeel, and Y. Gao. Mastering atari games with limited data. *Advances in Neural Information Processing Systems*, 34:25476–25488, Dec. 2021.

[6] T. Wang, X. Bao, I. Clavera, J. Hoang, Y. Wen, E. D. Langlois, S. Zhang, G. Zhang, P. Abbeel, and J. Ba. Benchmarking model-based reinforcement learning. *CoRR*, abs/1907.02057, 2019.

[7] N. A. Hansen, H. Su, and X. Wang. Temporal difference learning for model predictive control. In *Proceedings of the 39th International Conference on Machine Learning*, pages 8387–8406. PMLR, June 2022.

[8] N. Hansen, H. Su, and X. Wang. Td-mpc2: Scalable, robust world models for continuous control. In *The Twelfth International Conference on Learning Representations*, Oct. 2023.

[9] D. P. Bertsekas. Multiagent rollout algorithms and reinforcement learning. *CoRR*, abs/1910.00120, 2019.

[10] F. A. Oliehoek and C. Amato. *A Concise Introduction to Decentralized POMDPs*. SpringerBriefs in Intelligent Systems. Springer International Publishing, Cham, 2016. ISBN 978-3-319-28927-4 978-3-319-28929-8.

[11] G. E. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015.

[12] D. Hafner, J. Pasukonis, J. Ba, and T. P. Lillicrap. Mastering diverse domains through world models. *CoRR*, abs/2301.04104, 2023. doi:10.48550/ARXIV.2301.04104.

[13] T. Kipf, E. Fetaya, K.-C. Wang, M. Welling, and R. Zemel. Neural relational inference for interacting systems. In *Proceedings of the 35th International Conference on Machine Learning*, pages 2688–2697. PMLR, July 2018.

[14] X. B. Peng, Y. Guo, L. Halper, S. Levine, and S. Fidler. Ase: Large-scale reusable adversarial skill embeddings for physically simulated characters. *ACM Trans. Graph.*, 41(4):94:1–94:17, July 2022. ISSN 0730-0301. doi:10.1145/3528223.3530110.

[15] B. Balaji, P. Christodoulou, X. Lu, B. Jeon, and J. Bell-Masterson. Factoredrl: Leveraging factored graphs for deep reinforcement learning. *arXiv*, 2020.

[16] Z. Jiang, T. Zhang, M. Janner, Y. Li, T. Rocktäschel, E. Grefenstette, and Y. Tian. Efficient planning in a compact latent action space. In *The Eleventh International Conference on Learning Representations*, Sept. 2022.

[17] R. P. Anderson and D. Milutinović. Distributed path integral feedback control based on kalman smoothing for unicycle formations. In *2013 American Control Conference*, pages 4611–4616, June 2013. doi:10.1109/ACC.2013.6580550.

[18] C. Guestrin, D. Koller, and R. Parr. Multiagent planning with factored mdps. In *Advances in Neural Information Processing Systems*, volume 14. MIT Press, 2001.

[19] J. R. Kok and N. Vlassis. Collaborative multiagent reinforcement learning by payoff propagation. *Journal of Machine Learning Research*, 7(65):1789–1828, 2006. ISSN 1533-7928.

[20] Z. Liu, G. Zhou, J. He, T. Marcucci, L. Fei-Fei, J. Wu, and Y. Li. Model-based control with sparse neural dynamics. In *Thirty-Seventh Conference on Neural Information Processing Systems*, Nov. 2023.

[21] B. Peng, T. Rashid, C. A. S. de Witt, P.-A. Kamienny, P. H. S. Torr, W. Böhmer, and S. Whiteson. Facmac: Factored multi-agent centralised policy gradients, May 2021.

[22] Y. Shengchao, Y. Zhang, B. Zhang, J. Boedecker, and W. Burgard. Learning continuous control with geometric regularity from robot intrinsic symmetry. In *2024 IEEE International Conference on Robotics and Automation ICRA*, May 2024.

[23] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. F. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls, and T. Graepel. Value-decomposition networks for cooperative multi-agent learning based on team reward. In E. André, S. Koenig, M. Dastani, and G. Sukthankar, editors, *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018, Stockholm, Sweden, July 10-15, 2018*, pages 2085–2087. International Foundation for Autonomous Agents and Multiagent Systems Richland, SC, USA / ACM, 2018.

[24] T. Rashid, M. Samvelyan, C. S. de Witt, G. Farquhar, J. N. Foerster, and S. Whiteson. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In J. G. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 4292–4301. PMLR, 2018.

[25] K. Son, D. Kim, W. J. Kang, D. Hostallero, and Y. Yi. Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. *ArXiv*, May 2019.

[26] R. Bellman. A markovian decision process. *Journal of Mathematics and Mechanics*, 6(5): 679–684, 1957. ISSN 0095-9057.

[27] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou. Aggressive driving with model predictive path integral control. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1433–1440, May 2016. doi:10.1109/ICRA.2016.7487277.

[28] A. A. Rusu, S. G. Colmenarejo, Ç. Gülçehre, G. Desjardins, J. Kirkpatrick, R. Pascanu, V. Mnih, K. Kavukcuoglu, and R. Hadsell. Policy distillation. In Y. Bengio and Y. LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[29] L. Li, T. J. Walsh, and M. L. Littman. Towards a unified theory of state abstraction for mdps. In *International Symposium on Artificial Intelligence and Mathematics, AI&Math 2006, Fort Lauderdale, Florida, USA, January 4-6, 2006*, 2006.

[30] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, Vilamoura-Algarve, Portugal, Oct. 2012. IEEE. ISBN 978-1-4673-1736-8 978-1-4673-1737-5 978-1-4673-1735-1. doi:10.1109/IROS.2012.6386109.

[31] Z. Duan, Y. Zhang, S. Geng, G. Liu, J. Boedecker, and C. X. Lu. Fast ecot: Efficient embodied chain-of-thought via thoughts reuse. *arXiv*, 2025. doi:10.48550/ARXIV.2506.07639.

[32] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

A Additional Experimental Setups

A.1 Task Description

- **Reacher Three Easy**: A simple reaching task where a 3-link arm must move its end-effector to a target position with relatively slow dynamics.
- **Reacher Three Hard**: A more challenging version of the 3-link reaching task with faster dynamics or smaller target tolerance, requiring more precise control.
- **Hopper Hop**: A single-legged hopper must maintain balance and hop forward, testing stability and rhythmic control.
- **Fish Swim**: A simulated fish must propel itself forward and maneuver in water, involving complex body coordination.
- **Walker Walk**: A bipedal walker must move forward at a moderate speed, emphasizing stable gait generation.
- **Walker Run**: The walker must achieve faster locomotion while maintaining balance, increasing control difficulty.
- **Quadruped Walk**: A four-legged robot must walk steadily forward, testing coordination across multiple legs.
- **Quadruped Run**: A faster variant requiring the quadruped to run while keeping stability under high-speed conditions.

A.2 Hyperparameters

To compare all algorithms fairly, we set the model structures and hyperparameters equally. All algorithms are trained with Adam optimizer [32]. The full hyperparameters are shown in Table 2. All experiments are carried out on NVIDIA GeForce RTX 2080 Ti and Pytorch 1.10.1.

Table 2: Hyperparameters used in experiments.

Training	Values	Description
steps	10_000_00	Total number of training steps (gradient updates).
batch_size	512	Mini-batch size used for each optimization step.
reward_coef	0.5	Weight for reward prediction / reward-loss term in the training objective.
value_coef	0.1	Weight for value / critic loss term.
consistency_coef	0.0	Weight for any model-consistency loss (if used).
rho	0.5	Mixing / interpolation coefficient used in targets or blending (implementation-specific).
lr	1e-4	Base learning rate for optimizer.
enc_lr_scale	1.0	Multiplier applied to the encoder learning rate (encoder lr = lr $\times$ this).
grad_clip_norm	10	Maximum gradient norm for clipping.
tau	0.01	Soft-update coefficient for target networks (EMA factor).
discount	0.99	Discount factor.
buffer_size	1_000_000	Replay buffer capacity (number of transitions).
Planning (MPPI / MPC)		
mpc	true	Whether to enable model-predictive control (MPPI) at inference time.
iterations	6	Number of MPPI optimization iterations per control step.
num_samples	512	Number of trajectory samples drawn per iteration.
num_elites	64	Number of top samples (elites) used to update the sampling distribution.
num_pi_trajs	24	Number of policy (pi) trajectories evaluated (if using policy proposals).
horizon	3	Planning horizon (timesteps) for MPPI (comment shows prior value 5).
min_std	0.05	Minimum standard deviation for the sampling noise (stability floor).
max_std	2.0	Maximum standard deviation allowed for sampling noise.
temperature	0.5	Softmax/temperature parameter used when computing weights for samples.
Architecture		
num_enc_layers	2	Number of encoder (feedforward/conv) layers.
enc_dim	256	Width (hidden units) of encoder layers.
num_channels	32	Number of channels if using convolutional layers (per layer).
mlp_dim	512	Width of MLP heads / fully-connected layers.
latent_dim	50	Dimension of learned latent state per agent (or total, per design).
num_q	2	Number of Q-networks (ensemble) used for critic.
dropout	0.01	Dropout probability used in networks.