EfficientZero V2: Mastering Discrete and Continuous Control with Limited Data

Shengjie Wang [*123] **Shaohuai Liu** [*14] **Weirui Ye** [*123] **Jiacheng You** [1] **Yang Gao** [†123]

Abstract

Sample efficiency remains a crucial challenge in applying Reinforcement Learning (RL) to real-world tasks. While recent algorithms have made significant strides in improving sample efficiency, none have achieved consistently superior performance across diverse domains. In this paper, we introduce EfficientZero V2, a general framework designed for sample-efficient RL algorithms. We have expanded the performance of EfficientZero to multiple domains, encompassing both continuous and discrete actions, as well as visual and low-dimensional inputs. With a series of improvements we propose, EfficientZero V2 outperforms the current state-of-the-art (SOTA) by a significant margin in diverse tasks under the limited data setting. EfficientZero V2 exhibits a notable advancement over the prevailing general algorithm, DreamerV3, achieving superior outcomes in **50** of **66** evaluated tasks across diverse benchmarks, such as Atari 100k, Proprio Control, and Vision Control.

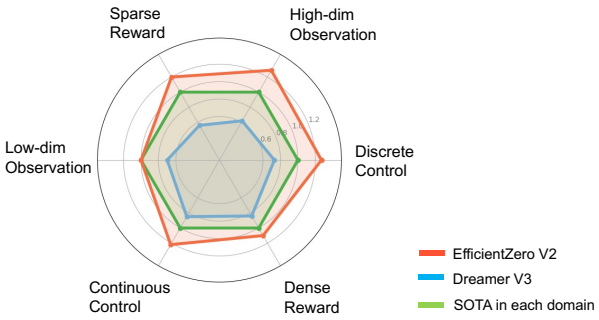

Figure 1. Comparison between EfficientZero V2, DreamerV3 and other SOTAs in each domain. We evaluate them under the Atari 100k, DMControl Proprio, and DMControl Vision benchmarks. We then set the performance of the previous SOTA as 1, allowing us to derive normalized mean scores for both EfficientZero V2 and Dreamer V3. EfficientZero V2 surpasses or closely matches the previous SOTA in each domain.

1. Introduction

Reinforcement learning (RL) has empowered computers to master diverse tasks, such as Go (Silver et al., 2018), video games (Ye et al., 2021), and robotics control (Hwangbo et al., 2019; Andrychowicz et al., 2020; Akkaya et al., 2019). However, these algorithms require extensive interactions with their environments, leading to significantly increased time and computational costs (Petrenko et al., 2023; Chen et al., 2022). For example, an RL-based controller requires nearly 100M interactions to reorient complex and diverse object shapes using vision input (Chen et al., 2023). Furthermore, building certain simulators for daily housework could be challenging. If gathering data in real-world settings, the process tends to be time-consuming and expensive. Consequently, it is crucial to explore and develop RL algorithms to achieve high-level performance with limited data.

Previous studies have introduced a range of algorithms aimed at enhancing sample efficiency, including TD-MPC series (Hansen et al., 2022; 2023), EfficientZero (Ye et al., 2021), and the Dreamer series (Hafner et al., 2019; **?**; 2023). Despite these advancements, these algorithms do not consistently attain superior sample efficiency across multiple domains. For example, TD-MPC (Hansen et al., 2022) leverages Model Predictive Path Integral (MPPI) (Rubinstein, 1997) control for planning. The huge computational burden in planning hinders its application in vision-based RL. EfficientZero (Ye et al., 2021) employs the Monte-Carlo Tree Search (MCTS) algorithm, which masters discrete control. However, EfficientZero is unable to handle high-dimensional action spaces, especially in continuous control scenarios. DreamerV3 (Hafner et al., 2023) is a universal algorithm that extends to diverse tasks in a wide range of domains. However, as shown in Fig. 1, DreamerV3 still has noticeable performance gaps to the state-of-the-art (SOTA) algorithms in each domain. Thus, there remains an open question on how to achieve both high performance and sample efficiency across various domains at the same time.

[*]Equal contribution [1]Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China [2]Shanghai Qi Zhi Institute, Shanghai, China [3]Shanghai Artificial Intelligence Laboratory, Shanghai, China [4]Texas A&M University. Correspondence to: Yang Gao <gaoyangiiis@mail.tsinghua.edu.cn>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

In this paper, we propose EfficientZero-v2 (EZ-V2), which can master tasks across various domains with superior sample efficiency. EZ-V2 successfully extends EfficientZero's strong performance to continuous control, demonstrating strong adaptability for diverse control scenarios. The main contributions of this work are as follows.

- We propose a general framework for sample-efficient RL. Specifically, it achieves consistent sample efficiency for discrete and continuous control, as well as visual and low-dimensional inputs.
- We evaluate our method in multiple benchmarks, outperforming the previous SOTA algorithms under limited data. As shown in Fig.1, the performance of EZ-V2 exceeds DreamerV3, a universal algorithm, by a large margin covering multiple domains with a data budget of 50k to 200k interactions.
- We design two key algorithmic enhancements: a sampled-based tree search for action planning, ensuring policy improvement in continuous action spaces, and a search-based value estimation strategy to more efficiently utilize previously gathered data and mitigate off-policy issues.

2. Related work

2.1. Sample Efficient RL

Sample efficiency in RL algorithms remains an essential direction for research. Inspired by advances in self-supervised learning, many RL algorithms now employ this approach to enhance the learning of representations from image inputs. For instance, CURL (Laskin et al., 2020) employs contrastive learning on hidden states to augment the efficacy of fundamental RL algorithms in image-based tasks. Similarly, SPR (Schwarzer et al., 2020) innovates with a temporal consistency loss combined with data augmentations, resulting in enhanced performance.

Furthermore, Model-Based Reinforcement Learning (MBRL) has demonstrated high sample efficiency and notable performance in both discrete and continuous control domains. SimPLE (Kaiser et al., 2019), by modeling the environment, predicts future trajectories, thereby achieving commendable performance in Atari games with limited data. TD-MPC (Hansen et al., 2022) utilizes data-driven Model Predictive Control (MPC) (Rubinstein, 1997) with a latent dynamics model and a terminal value function, optimizing trajectories through short-term planning and estimating long-term returns. The subsequent work, TD-MPC2 (Hansen et al., 2023), excels in multi-task environments. The TD-MPC series employs MPC to generate imagined latent states for action planning. In contrast, our method employs a more efficient action planning module called Sampling-based Gumbel Search, leading to lower computational costs. Dreamer (Hafner et al., 2019), a reinforcement learning agent, develops behaviors from predictions within a compact latent space of a world model. Its latest iteration, Dreamer V3 (Hafner et al., 2023), is a general algorithm that leverages world models and surpasses previous methods across a wide range of domains. It showcases its sample efficiency by learning online and directly in real-world settings (Wu et al., 2022). Although long-horizon planning in Dreamer V3 (Hafner et al., 2023) enhances the quality of the collected data, an imagination horizon of $H = 15$ may be excessively long, potentially leading to an accumulation of model errors.

2.2. MCTS-based Work

AlphaGo (Silver et al., 2016) is the first algorithm to defeat a professional human player in the game of Go, utilizing Monte-Carlo Tree Search (MCTS) (Coulom, 2006) along with deep neural networks. AlphaZero (Silver et al., 2017) extends this approach to additional board games such as Chess and Shogi. MuZero (Schrittwieser et al., 2020), aspiring to master complex games without prior knowledge of their rules, learns to predict game dynamics by training an environment model. Building upon MuZero, EfficientZero (Ye et al., 2021) achieves superhuman performance in Atari games with only two hours of real-time gameplay, attributed to the self-supervision of the environment model. However, applying MuZero to tasks with large action spaces significantly increases the computational cost of MCTS due to the growing number of simulations. Gumbel MuZero (Danihelka et al., 2021) effectively diminishes the complexity of search within vast action spaces by implementing Gumbel search, although it does not extend to continuous action domains. Sample MuZero (Hubert et al., 2021) proposes a sampling-based MCTS that contemplates subsets of sampled actions, thus adapting the MuZero framework for continuous control. Recent developments have also seen MuZero applied in stochastic environments (Antonoglou et al., 2021) and its value learning augmented by path consistency (PC) optimality regularization (Zhao et al., 2022). Our method notably enhances Gumbel search for continuous control and requires only half the number of search simulations compared to Sample MuZero (Hubert et al., 2021).

3. Preliminary

3.1. Reinforcement Learning

Reinforcement Learning (RL) can be formulated as a Markov Decision Process (MDP) (Bellman, 1957). An MDP in this context is formalized as a tuple (S, A, T, R, γ), where $s \in S$ represents states, $a \in A$ denotes actions, $T : S \times A \to S$ is the transition function, $R : S \times A \to \mathbb{R}$

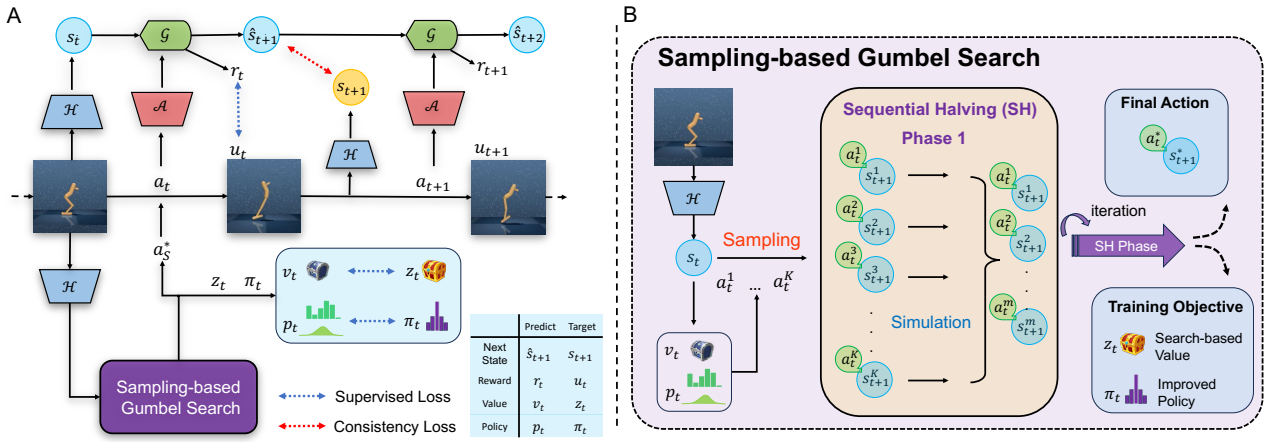

Figure 2. Framework of EZ-V2. (**A**) How EZ-V2 trains its model. The representation $\mathcal{H}$ takes observations as inputs and outputs the state. The dynamic model $\mathcal{G}$ predicts the next state and reward based on the current state and action. Sampling-based Gumbel search outputs the target policy π_t and target value z_t. (**B**): How the sampling-based Gumbel search uses the model to plan. The process contains action sampling and selection. The iterative action selection outputs the recommended action a_S^*, search-based value target (target value), and improved policy (target policy).

is the reward function associated with a particular task, and γ is the discount factor. The goal in RL is to find an optimal policy π^* that maximizes the expected sum of discounted rewards over time, formally expressed as $\max_\pi \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right]$, where a_t is an action taken according to the policy π at state s_t. In this work, to improve sample efficiency, we learn a model of the environment during training. Meanwhile, planning with the learned model makes action selection more efficient.

3.2. Gumbel-Top-k Trick

The Gumbel-Top-k trick (Kool et al., 2019) can choose the top n actions without replacement in a categorical distribution π. Specifically, the action sample A can be sampled by the Gumbel-Max trick, which is defined as follows.

$$\begin{aligned} \left(g \in \mathbb{R}^k\right) &\sim \text{Gumbel}(0) \\ A &= \arg\max_a (g(a) + \text{logits}(a)) \end{aligned} \tag{1}$$

where $\text{logits}(a)$ is the logit of the action a, and g is a vector of k Gumbel variables. Hereafter, we can sample n actions without replacement.

$$\begin{aligned} A_1 &= \arg\max_a (g(a) + \text{logits}(a)) \\ &\vdots \\ A_n &= \arg\max_{a \notin \{A_1, \ldots, A_{n-1}\}} (g(a) + \text{logits}(a)). \end{aligned} \tag{2}$$

Furthermore, we denote the set of n top actions by $\text{argtop}(g + logits, n) = A_1, A_2, ..., A_n$.

3.3. EfficientZero

3.3.1. Newtork Structure

EfficientZero algorithm learns a predictive model in a latent space and then performs planning over actions using this model. Specifically, the components of EfficientZero consist of a representation function, a dynamic function, a policy function, and a value function, which are formulated as follows.

- Representation Function: $\mathcal{H} : s_t = \mathcal{H}(o_t)$
- Dynamic Function: $\mathcal{G} : \hat{s}_{t+1}, r_t = \mathcal{G}(s_t, a_t)$
- Policy Function: $\mathcal{P} : p_t = \mathcal{P}(s_t)$
- Value Function : $\mathcal{V} : v_t = \mathcal{V}(s_t)$

Among these components, o_t represents the current observation, s_t is the current latent state, and $\hat{s}_{t+1}$ is the predicted next state. The representation function learns a compact state representation of the input o_t. The dynamic function predicts the next state $\hat{s}_{t+1}$ and the reward r_t. The policy function outputs the current policy p_t, and the value function provides the value estimation v_t at the current state. All components are implemented as neural networks. EZ-V2 uses a similar network structure (see Figure 2), with details of the architecture for each component provided in Appendix G. The main difference is that the action embedding block $\mathcal{A}$ encodes the action a_t as a latent vector within the dynamic function.

3.3.2. TRAINING PROCESS

For the dynamic function, EfficientZero employs a supervised learning method using the true reward u_t. Furthermore, EfficientZero introduces temporal consistency (Ye et al., 2021), which strengthens the supervision between the predicted next state $\hat{s}_{t+1}$ and the true next state s_{t+1}. Using the learned model, an MCTS method is used for planning actions. The method can generate the target policy π_t and target value z_t for further supervised learning of the policy and value functions. Simultaneously, it outputs the recommended action a_S^* for interaction with the environment. The iterative process of interaction and training enhances the accuracy of the dynamic function's predictions and gradually improves the policy and value functions. EZ-V2 inherits the training process of EfficientZero but replaces the MCTS method with a sampling-based Gumbel search, ensuring policy improvement in continuous action spaces. Figure 2 (A) intuitively illustrates the training process of EZ-V2.

All parameters of the components are trained jointly to match the target policy, value, and reward.

$$\begin{aligned}\mathcal{L}_t = \lambda_1 \mathcal{L}_{\mathcal{R}}(u_t, r_t) + \lambda_2 \mathcal{L}_{\mathcal{P}}(\pi_t, p_t) \\ + \lambda_3 \mathcal{L}_{\mathcal{V}}(z_t, v_t) + \lambda_4 \mathcal{L}_{\mathcal{G}}(s_{t+1}, \hat{s}_{t+1})\end{aligned} \tag{3}$$

where u_t denotes the environmental reward, π_t is the target policy from the search and z_t represents the target value from the search. $\mathcal{L}_{\mathcal{R}}$, $\mathcal{L}_{\mathcal{P}}$ and $\mathcal{L}_{\mathcal{V}}$ all represent supervised learning losses. $\mathcal{L}_{\mathcal{G}}$ is the temporal consistency loss (Schwarzer et al., 2020), which enhances the similarity between the predicted next state $\hat{s}_{t+1}$ with the next state s_{t+1} and is formulated as:

$$\mathcal{L}_{\mathcal{G}}\left(s_{t+1}, \hat{s}_{t+1}\right) = \mathcal{L}_{cos}\left(sg\left(P_1\left(s_{t+1}\right)\right), P_2\left(P_1\left(\hat{s}_{t+1}\right)\right)\right) \tag{4}$$

where $\mathcal{L}_{cos}$ is the negative cosine similarity loss and sg means the stop gradient operator. The asymmetric design of employing P_1 and P_2 follows the setting in SimSiam (Chen & He, 2021). More details about the architecture also can be found in EfficientZero (Ye et al., 2021). To enhance the prediction accuracy of the model, we unroll the loss with l_{unroll} steps to train the model.

$$\mathcal{L} = \frac{1}{l_{\text{unroll}}} \sum_{i=0}^{l_{\text{unroll}}-1} \mathcal{L}_{t+i} \tag{5}$$

More details of the training pipeline can be found in Appendix H.

4. Method

4.1. Overview

EZ-V2 is built upon EfficientZero, a model-based algorithm that performs planning using MCTS within a learned environment model. EZ-V2 successfully extends EfficientZero's high sample efficiency to various domains. To realize this extension, EZ-V2 addresses two pivotal questions:

- *How to perform efficient planning using tree search in high-dimensional and continuous action spaces?*
- *How to further strengthen the ability to utilize stale transitions under limited data?*

Specifically, we propose a series of improvements. We construct a sampling-based tree search for policy improvement in continuous control. Furthermore, we propose a search-based value estimation method to alleviate off-policy issues in replaying stale interaction data. The differences compared to EfficientZero are detailed in Appendix A.

4.2. Policy Learning with Tree Search

The policy learning in EZ-V2 consists of two stages: (i) *obtaining the target policy from tree search* and (ii) *supervised learning using the target policy*. The tree search method we propose guarantees policy improvement as defined in Definition 4.1 and enhances the efficiency of exploration in a continuous action space. The training objective aims to refine the policy function by aligning it with the target policy obtained from the tree search.

Definition 4.1 (Policy Improvement). *A planning method over actions satisfies policy improvement if the following inequality holds at any given state s.*

$$q(s, a_S^*) \geq \mathbb{E}_{a \sim p_t}[q(s, a)] \tag{6}$$

where a_S^* is the recommended action from the planning method, p_t is the current policy, and q is the Q-value function with respect to p_t.

4.2.1. TARGET POLICY FROM TREE SEARCH

In this paper, we choose the tree search method as the improvement operator, which can construct a locally superior policy over actions (policy improvement) based on a learned model. Each node of the search tree is associated with a state s, and an edge is denoted as (s, a). The tree stores the estimated Q-value of each node and updates it through simulations. Finally, we select an action that strikes a balance between exploitation and exploration, based on the Q-value.

More specifically, the basic tree search method we adopt is the Gumbel search (Danihelka et al., 2021). This method is recognized for its efficiency in tree searching and its guarantee of policy improvement. At the onset of a search process, the Gumbel search samples K actions using the Gumbel-Top-k trick (Section 3.2). It then approaches the root action selection as a bandit problem, aiming to choose the action with the highest Q-value. To evaluate the set of sampled actions, we employ a bandit algorithm known

as Sequential Halving (Karnin et al., 2013). However, the Gumbel search primarily investigates planning in discrete action spaces.

To support high-dimensional continuous control, we design a sampling-based Gumbel search, as depicted in Fig. 2 (B). Given the challenges posed by high-dimensional and large continuous action spaces, striking a balance between exploration and exploitation is crucial for the performance of tree search, especially when using a limited number of sampled actions. To address this challenge, we propose an action sampling method that not only achieves excellent exploration capabilities but also ensures that the search method satisfies policy improvement as defined in Definition 4.1.

Our method is implemented as follows. Given any state s, we sample K actions. A portion of these actions originates from the current policy p_t, while another portion is sampled from a prior distribution p'_t. We denote the complete action set as $A_S = [A_{S1}, A_{S2}]$, where A_{S1} and A_{S2} represent the two portions, respectively. This design enhances exploration because A_{S2} can introduce actions that have a low prior under the current policy p_t. The bandit process then selects the action a^*_S from the action set A_S with the highest $q(s,a)$, expressed as $a^*_S = \arg\max_{a \in A_S}(q(s,a))$. This process guarantees policy improvement as outlined in Definition 4.1, because:

$$\begin{aligned} q(s, a^*_S) &\geq \max\left(\frac{\sum_{a\in A_{S1}} q(s,a)}{|A_{S1}|}, \frac{\sum_{a\in A_{S2}} q(s,a)}{|A_{S2}|}\right) \\ &\geq \frac{\sum_{a\in A_{S1}} q(s,a)}{|A_{S1}|} \\ &= \mathbb{E}_{a\sim p_t}[q(s,a)], \ \text{ as } \ |A_{S1}| \to \infty \end{aligned} \tag{7}$$

where $|A_{S1}|$ represents the number of actions in A_{S1}. The first line holds because a^*_S is the action with the highest $q(s,a)$ in A_S and $[A_{S1}, A_{S2}] = A_S$. We apply the law of large numbers to transition from line 2 to 3. In practical implementation, we model the current policy p_t as a Gaussian distribution, and the sampling distribution p'_t is a flattened version of the current policy. Our experiments demonstrate that this design facilitates exploration in continuous control.

For action sampling at non-root nodes, we modify the sampling method to reduce the variance in the estimation of Q-values. Specifically, actions at non-root nodes are sampled solely from the current policy p_t. Additionally, the number of actions sampled at non-root nodes is fewer than those at the root node. This reduction in the number of sampled actions at non-root nodes facilitates an increase in search depth, thereby avoiding redundant simulations on similar sampled actions.

Upon completing the sampling-based Gumbel search, we obtain the target policy. We construct two types of target policies. The first is the recommended action a^*_S obtained from the tree search. In addition to a^*_S, the search also yields $q(s,a)$ for the visited actions. To smooth the target policy, we build the second type as a probability distribution based on the Q-values of root actions (for more details, see Appendix B).

4.2.2. Learning using Target Policy

In this part of the process, we distill the target policy π_t into the learnable policy function p_t. We aim to minimize the cross-entropy between p_t and the target policy π_t:

$$\mathcal{L}_{\mathcal{P}}(p_t, \pi_t) = \mathbb{E}_{a\sim\pi_t}\left[-\log p_t(a)\right] \tag{8}$$

Additionally, in high-dimensional action spaces, we utilize the other target a^*_S:

$$\mathcal{L}_{\mathcal{P}}(p_t, a^*_S) = -\log p_t\left(a^*_S\right) \tag{9}$$

Compared with Equation (8), Equation (9) facilitates early exploitation in tasks with a large action dimension, such as the 'Quadruped walk' in DM Control (Tassa et al., 2018). We provide an intuitive example to illustrate its advantage in Appendix C.

4.3. Search-based Value Estimation

Improving the ability to utilize off-policy data is crucial for sample-efficient RL. Sample-efficient RL algorithms often undergo drastic policy shifts within limited interactions, leading to estimation errors for early-stage transitions in conventional methods, such as N-step bootstrapping and TD-λ. EfficientZero proposed an adaptive step bootstrapping method to alleviate the off-policy issue. However, this method utilizes the multi-step discount sum of rewards from an old policy, which can lead to inferior performance. Therefore, it is essential to enhance value estimation to better utilize stale transitions.

We propose leveraging the current policy and model to conduct value estimation, which we term **Search-Based Value Estimation (SVE)**. The expanding search tree generates imagined trajectories that provide bootstrapped samples for root value estimations. We now use the mean of these empirical estimations as target values. Notably, this value estimation method can be implemented within the same process as the policy reanalysis proposed by MuZero (Schrittwieser et al., 2021), thereby not introducing additional computational overhead. The mathematical definition of SVE is as follows.

Definition 4.2 (Search-Based Value Estimation). *Using imagined states and rewards $\hat{s}_{t+1}, \hat{r}_t = \mathcal{G}(\hat{s}_t, \hat{a}_t)$ obtained from our learnable dynamic function, the value estimation of a given state s_0 can be derived from the empirical mean*

of N bootstrapped estimations, which is formulated as

$$\hat{V}_{\mathrm{S}}(s_0) = \frac{\sum_{n=0}^{N} \hat{V}_n(s_0)}{N} \tag{10}$$

where N denotes the number of simulations, $\hat{V}_n(s_0)$ is the bootstrapped estimation of the n-th node expansion, which is formulated as

$$\hat{V}_n(s_0) = \sum_{t=0}^{H(n)} \gamma^t \hat{r}_t + \gamma^{H(n)} \hat{V}(\hat{s}_{H(n)}) \tag{11}$$

where $H(n)$ denotes the search depth of the n-th iteration.

Through the imagined search process with the newest policy and model, SVE provides a more accurate value estimation for off-policy data. Furthermore, investigating the nature of estimation errors is critical. We derive an upper bound for the value estimation error, taking into account model errors, as illustrated in Theorem 4.3.

Corollary 4.3 (Search-Based Value Estimation Error). *Define s_t, a_t, r_t to be the states, actions, and rewards resulting from current policy π using true dynamics $\mathcal{G}^*$ and reward function $\mathcal{R}^*$, starting from $s_0 \sim \nu$ and similarly define $\hat{s}_t, \hat{a}_t, \hat{r}_t$ using learned function $\mathcal{G}$. Let reward function $\mathcal{R}$ to be $L_r-Lipschitz$ and value function $\mathcal{V}$ as $L_V-Lipschitz$. Assume $\epsilon_s, \epsilon_r, \epsilon_v$ as upper bounds of state transition, reward, and value estimations respectively. We define the error bounds of each estimation as*

$$\max_{n\in[N],t\in[H(n)]} \mathbb{E}\left[\|\hat{s}_t - s_t\|^2\right] \leq \epsilon_s^2 \tag{12}$$

$$\max_{n\in[N],t\in[H(n)]} \mathbb{E}\left[\|\mathcal{R}(s_t) - \mathcal{R}^*(s_t)\|^2\right] \leq \epsilon_r^2 \tag{13}$$

$$\max_{n\in[N],t\in[H(n)]} \mathbb{E}\left[\|\mathcal{V}(s_t) - \mathcal{V}^*(s_t)\|^2\right] \leq \epsilon_v^2 \tag{14}$$

within a tree-search process. Then we have errors

$$MSE_\nu(\hat{V}_S) \leq \frac{4}{N^2} \sum_{n=0}^{N} \left(\sum_{t=0}^{H(n)} \gamma^{2t}(L_r^2\epsilon_s^2 + \epsilon_r^2) + \gamma^{2H(n)}(L_V^2\epsilon_s^2 + \epsilon_v^2) \right) \tag{15}$$

where N is the simulation number of the search process. $H(n)$ denotes the depth of the n-th search iteration.

The detailed proof can be found in Appendix E.1. SVE possesses several advantageous properties, such as a convergent series coefficient and bounded model errors. Intuitively, the estimation error bound will converge to 0 when the dynamic function approaches optimality, denoted as $\epsilon \to 0$.

Theorem 4.3 shows that model inaccuracies can amplify SVE's estimation error, especially in the early training stages or when sampling fresh transitions. To address this, we introduce a mixed value target, combining multi-step TD-targets for early training and fresh experience sampling. The mixed target is defined as:

$$V_{mix} = \begin{cases} \sum_{i=0}^{l-1} \gamma^i u_{t+i} + \gamma^l v_{t+l}, & \text{if } i_t < T_1 \\ & \text{or } i_s > |D| - T_2 \\ \hat{V}_{\mathrm{S}}, & \text{otherwise} \end{cases} \tag{16}$$

Here, l is the horizon for the multi-step TD method. The variable i_t denotes the current training step, while T_1 refers to the initial steps. The term i_s indicates the sampling index from the buffer. The buffer size is represented by $|D|$, and T_2 is the designated threshold for assessing the staleness of data. More details can be found in Appendix F.

5. Experiment

In this section, we aim to evaluate the general sample efficiency of EZ-V2 on a total of 66 diverse tasks. The tasks include scenarios with low and high-dimensional observation, discrete and continuous action spaces, and dense and sparse rewards. We then present an ablation study on the sampling-based Gumbel search and mixed value target we propose.[1]

5.1. Experimental Setup

To assess sample efficiency, we measure algorithm performance with limited environment steps. In discrete control, we use the **Atari 100k** benchmark (Brockman et al., 2016), encompassing 26 Atari games and limiting training to 400k environment steps, equivalent to 100k steps with action repeats of 4.

For continuous control evaluation, we utilize the DeepMind Control Suite (DMControl; (Tassa et al., 2018)), comprising various tasks in classical control, locomotion, and manipulations. Referring to the categorizations in Sample MuZero (Hubert et al., 2021), tasks are divided into **easy** and **hard** categories. The easy tasks use half the interaction data of the hard tasks. We establish the following benchmarks:

- **Proprio Control 50k** for easy tasks with low-dimensional state inputs.
- **Proprio Control 100k** for hard tasks with low-dimensional state inputs.
- **Vision Control 100k** for easy tasks with image observations.
- **Vision Control 200k** for hard tasks with image observations.

[1]The code is released at https://github.com/Shengjiewang-Jason/EfficientZeroV2.

Table 1. Scores achieved on the Atari 100k benchmark indicate that EZ-V2 achieves super-human performance within just 2 hours of real-time gameplay. Our method surpasses the previous state-of-the-art, EfficientZero. The results for Random, Human, SimPLe, CURL, DrQ, SPR, MuZero, and EfficientZero are sourced from (Ye et al., 2021).

Game	Random	Human	SimPLe	CURL	DrQ	SPR	MuZero	EfficientZero	DreamerV3	BBF	**EZ-V2 (Ours)**
Alien	227.8	7127.7	616.9	558.2	771.2	801.5	530.0	808.5	959	1173.2	**1557.7**
Amidar	5.8	1719.5	88.0	142.1	102.8	176.3	38.8	148.6	139	**244.6**	184.9
Assault	222.4	742.0	527.2	600.6	452.4	571.0	500.1	1263.1	706	**2098.5**	1757.5
Asterix	210.0	8503.3	1128.3	734.5	603.5	977.8	1734.0	25557.8	932	3946.1	**61810.0**
Bank Heist	14.2	753.1	34.2	131.6	168.9	380.9	192.5	351.0	649	732.9	**1316.7**
BattleZone	2360.0	37187.5	5184.4	14870.0	12954.0	16651.0	7687.5	13871.2	12250	**24459.8**	14433.3
Boxing	0.1	12.1	9.1	1.2	6.0	35.8	15.1	52.7	52.7	**85.8**	75.0
Breakout	1.7	30.5	16.4	4.9	16.1	17.1	48.0	**414.1**	31	370.6	400.1
ChopperCmd	811.0	7387.8	1246.9	1058.5	780.3	974.8	1350.0	1117.3	420	**7549.3**	1196.6
Crazy Climber	10780.5	35829.4	62583.6	12146.5	20516.5	42923.6	56937.0	83940.2	97190	58431.8	**112363.3**
Demon Attack	152.1	1971.0	208.1	817.6	1113.4	545.2	3527.0	13003.9	303	13341.4	**22773.5**
Freeway	0.0	29.6	20.3	**26.7**	9.8	24.4	21.8	21.8	0	25.5	0.0
Frostbite	65.2	4334.7	254.7	1181.3	331.1	1821.5	255.0	296.3	909	**2384.8**	1136.3
Gopher	257.6	2412.5	771.0	669.3	636.3	715.2	1256.0	3260.3	3730	1331.2	**3868.7**
Hero	1027.0	30826.4	2656.6	6279.3	3736.3	7019.2	3095.0	9315.9	**11161**	7818.6	9705.0
Jamesbond	29.0	302.8	125.3	471.0	236.0	365.4	87.5	517.0	445	**1129.6**	468.3
Kangaroo	52.0	3035.0	323.1	872.5	940.6	3276.4	62.5	724.1	4098	**6614.7**	1886.7
Krull	1598.0	2665.5	4539.9	4229.6	4018.1	3688.9	4890.8	5663.3	7782	8223.4	**9080.0**
Kung Fu Master	258.5	22736.3	17257.2	14307.8	9111.0	13192.7	18813.0	**30944.8**	21420	18991.7	28883.3
Ms Pacman	307.3	6951.6	1480.0	1465.5	960.5	1313.2	1265.6	1281.2	1327	2008.3	**2251.0**
Pong	-20.7	14.6	12.8	-16.5	-8.5	-5.9	-6.7	20.1	18	16.7	**20.8**
Private Eye	24.9	69571.3	58.3	218.4	-13.6	124.0	56.3	96.7	**882**	40.5	99.8
Qbert	163.9	13455.0	1288.8	1042.4	854.4	669.1	3952.0	13781.9	3405	4447.1	**16058.3**
Road Runner	11.5	7845.0	5640.6	5661.0	8895.1	14220.5	2500.0	17751.3	15565	**33426.8**	27516.7
Seaquest	68.4	42054.7	683.3	384.5	301.2	583.1	208.0	1100.2	618	1232.5	**1974.0**
Up N Down	533.4	11693.2	3350.3	2955.2	3180.8	**28138.5**	2896.9	17264.2	-	12101.7	15224.3
Normed Mean	0.000	1.000	0.443	0.381	0.357	0.704	0.562	1.945	1.120	2.247	**2.428**
Normed Median	0.000	1.000	0.144	0.175	0.268	0.415	0.227	1.090	0.490	0.917	**1.286**

Each benchmark includes 10 tasks. Action repeats are set to 2 and the maximum episode length is 1000 for all 4 benchmarks, in line with previous studies (Hafner et al., 2023; Hansen et al., 2023).

We choose strong baselines for each domain, which include **SAC** (Haarnoja et al., 2018), **DrQ-v2** (Yarats et al., 2021), **TD-MPC2** (Hansen et al., 2023), **DreamerV3** (Hafner et al., 2023), **EfficientZero** (Ye et al., 2021), and **BBF** (Schwarzer et al., 2023). For more details on the implementation, please refer to Appendix I.

5.2. Comparison with Baselines

Atari 100k: The performance of EZ-V2 on the Atari 100k benchmark is elaborated in Table 1. When scores are normalized against those of human players, EZ-V2 attains a mean score of 2.428 and a median score of 1.286, surpassing the previous state-of-the-art, BBF (Schwarzer et al., 2023) and EfficientZero (Ye et al., 2021). In contrast to BBF, our method employs fewer network parameters and a lower replay ratio. Such enhancements in performance and computational efficiency are attributed to the learning of the environment model and the implementation of Gumbel search in action planning. Moreover, EZ-V2 necessitates fewer search simulations compared to EfficientZero and still manages to achieve superior performance. The utilization of a mixed value target decisively mitigates off-policy issues associated with using outdated data, marking a substantial advancement beyond the adaptive step bootstrapping target used by EfficientZero.

Proprio Control: The results in Table 2 showcase that our method achieves a mean score of 723.2 across 20 tasks with limited data. While the performance of the current state-of-the-art, TD-MPC2, is comparable to that of EZ-V2, our method achieves faster inference times. TD-MPC2's planning with MPPI involves predicting 9216 latent states to attain similar performance levels. In contrast, EZ-V2's tree search-based planning only utilizes 32 imagined latent states, resulting in lighter computational demands.

Vision Control: As shown in Table 2, our method achieves a mean score of 726.1, surpassing the previous state-of-the-art, DreamerV3, by 45%. Notably, it sets new records in 16 out of 20 tasks. Furthermore, our method demonstrates significant improvements in tasks with sparse rewards, as shown in Fig. 7. For instance, in the 'Cartpole-Swingup-Sparse' task, our method scores 763.6 compared to DreamerV3's 392.4. This substantial progress is attributed to two key algorithmic modifications: the planning with tree search, which ensures policy improvement and offers excellent exploratory capabilities, and the mixed value target, which enhances the accuracy of value learning, especially with stale data.

As a general and sample-efficient RL framework, EZ-V2

Table 2. Scores achieved on the Proprio Control 50/100k and Vision Control 100/200k benchmarks (with 3 seeds run for each) demonstrate that EZ-V2 consistently maintains sample efficiency, whether with proprioceptive or visual inputs. The tasks are categorized into easy and hard groups as proposed by (Hubert et al., 2021). The results of DreamerV3 are sourced from the official data (Hafner et al., 2023).

Benchmark	Proprio Control 50k				Vision Control 100k			
Task	SAC	TD-MPC2	DreamerV3	**EZ-V2 (Ours)**	CURL	DrQ-v2	DreamerV3	**EZ-V2 (Ours)**
Cartpole Balance	**997.6**	962.8	839.6	947.3	963.3	**965.5**	956.4	911.7
Cartpole Balance Sparse	993.1	942.8	559.0	**999.2**	999.4	**1000.0**	813.0	951.5
Cartpole Swingup	**861.6**	826.7	527.7	805.4	**765.4**	756.0	374.8	747.8
Cup Catch	949.9	**976.0**	729.6	969.8	932.3	468.0	947.7	**954.7**
Finger Spin	900.0	**965.8**	765.8	837.1	850.2	459.4	633.2	**927.6**
Pendulum Swingup	158.9	520.1	**830.4**	825.4	144.1	233.3	619.3	**726.7**
Reacher Easy	744.0	903.5	693.4	**940.3**	467.9	722.1	441.4	**946.3**
Reacher Hard	646.5	580.4	768.0	**795.4**	112.7	202.9	120.4	**961.5**
Walker Stand	870.0	**973.9**	767.3	953.6	733.8	426.1	939.5	**944.9**
Walker Walk	813.2	**965.5**	475.2	944.0	538.5	681.5	771.2	**888.8**
	Proprio Control 100k				Vision Control 200k			
Acrobot Swingup	44.1	**303.2**	62.8	297.7	6.8	15.1	67.4	**231.8**
Cartpole Swingup Sparse	256.6	421.4	172.7	**795.4**	8.8	81.2	392.4	**763.6**
Cheetah Run	**680.9**	614.4	400.8	677.8	405.1	418.4	587.3	**631.6**
Finger Turn Easy	630.8	**793.3**	560.5	310.7	371.5	286.8	366.6	**799.2**
Finger Turn Hard	414.0	**604.8**	474.2	374.1	236.3	268.4	258.5	**794.6**
Hopper Hop	0.1	84.5	9.7	**186.5**	84.5	26.3	76.3	**206.4**
Hopper Stand	3.8	**807.9**	296.1	795.4	627.7	290.2	652.5	**805.7**
Quadruped Run	139.7	**742.1**	289.0	510.6	170.9	339.4	168.0	**384.8**
Quadruped Walk	237.5	853.7	256.2	**925.8**	131.8	311.6	122.6	**433.3**
Walker Run	635.4	**780.5**	478.9	657.2	274.7	359.9	**618.2**	475.3
Mean	552.0	**740.9**	517.1	723.2	437.3	410.3	498.5	**726.1**
Median	633.3	**806.4**	543.4	800.4	324.9	330.6	484.5	**788.1**

consistently demonstrates high sample efficiency in tasks featuring low and high-dimensional observations, discrete and continuous action spaces, and both dense and sparse reward structures. Detailed training curves can be found in Appendix J.

5.3. Ablation Study

In this section, we discuss the effectiveness of the main modifications: the sampling-based Gumbel search and the mixed value target.

Ablations of Search: The comparative analysis between our search method and Sample MCTS is illustrated in Fig. 3. Sample MCTS, a tree search technique tailored for continuous control developed by Sample MuZero (Hubert et al., 2021), is depicted in green. Our search method, highlighted in red, exhibits superior performance. Whereas Sample MCTS necessitates $n = 50$ simulations, our approach significantly reduces the computational burden to merely 32 simulations.

Further, we display performance curves for 16 and 8 simulations. As demonstrated in Fig. 3, an increased number of simulations enhances performance in complex tasks such as 'Quadruped Walk', and notably, our method with merely 8 simulations surpasses Sample MCTS. This shows that the sampling-based Gumbel search achieves a superior balance between exploration and exploitation, backed by guaranteed

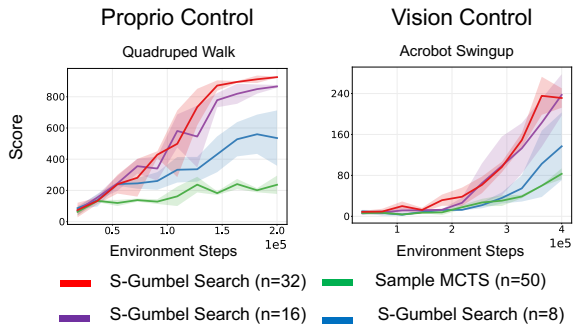

Figure 3. Ablation study of our search method, namely the sampling-based Gumbel search (S-Gumbel search). We compare it with our search method with different numbers of simulations (n=16, 8) and Sample MCTS (Hubert et al., 2021). Our method outperforms Sample MCTS, and increasing the number of simulations improves our method's performance on hard tasks.

policy improvement. Additional results for other tasks are provided in Appendix J.2.

Ablations of Value Targets. It can be seen in Fig. 4 that our method (colored red) alleviates the off-policy issue compared with the multi-step TD target (colored purple), thus achieving better performance. Furthermore, we also compare with the value target z_t derived from the optimal-Q Bellman equation, shown as follows:

$$z_t = u_t + \gamma q(s_{t+1}, a_{t+1}), \quad a_{t+1} \sim p_t(a|s_{t+1}) \tag{17}$$

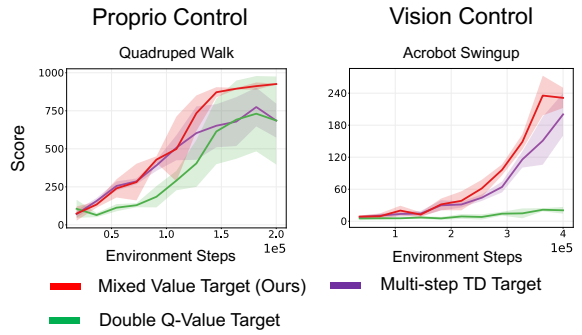

Figure 4. Ablation study of our value target, known as the mixed value target. We compare it with different value targets, including the multi-step TD target and the double Q-value target. The mixed value target consistently achieves high performance in both Proprio Control and Vision Control tasks.

This technique also addresses the off-policy issue as the value target is based on the optimal Q-value estimation. Meanwhile, we employ the double Q-head trick. This estimation method is denoted as double Q-value target in Fig. 4. Although the double Q-value target (colored green) also avoids the off-policy issue, the experiments illustrate that our method (colored red) exhibits consistent and robust performance across tasks. This is because our method matches the true value more rapidly by utilizing multi-step predicted rewards during the tree search. Additional curves related to value ablation experiments can be found in Appendix J.2.

Furthermore, practical comparisons between the TD-MPC2 and EZ-V2 algorithms in terms of computational load are provided in Appendix J.3.

6. Conclusion

This paper presents EfficienctZero V2 (EZ-V2), a general framework for sample efficient RL. EZ-V2, which is built upon EfficientZero, extends to master continuous control. Furthermore, EZ-V2 achieves both superior performance and sample efficiency in tasks with visual and low-dimensional inputs. More specifically, EZ-V2 outperforms Dreamer V3 by a large margin across various types of domains, including Atari 100k, Proprio Control, and Vision Control benchmarks. Moreover, we evaluate the performance of EZ-V2 without conditions of limited data. EZ-V2 outperforms or is comparable to baselines with more interaction data, though the performance gap between EZ-V2 and other algorithms narrows as the amount of interaction data increases. Due to the high sample efficiency, we will broaden our evaluation of EZ-V2 across a wider variety of benchmarks, especially in real-world tasks like robotic manipulation. The superior sample efficiency of EZ-V2 holds great promise for enhancing online learning in real-world robotic scenarios.

Acknowledgements

This work is supported by the Ministry of Science and Technology of the People's Republic of China, the 2030 Innovation Megaprojects "Program on New Generation Artificial Intelligence" (Grant No. 2021AAA0150000). This work is also supported by the National Key R&D Program of China (2022ZD0161700).

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Akkaya, I., Andrychowicz, M., Chociej, M., Litwin, M., McGrew, B., Petron, A., Paino, A., Plappert, M., Powell, G., Ribas, R., et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

Andrychowicz, O. M., Baker, B., Chociej, M., Jozefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

Antonoglou, I., Schrittwieser, J., Ozair, S., Hubert, T. K., and Silver, D. Planning in stochastic environments with a learned model. In *International Conference on Learning Representations*, 2021.

Bellman, R. A markovian decision process. *Journal of mathematics and mechanics*, pp. 679–684, 1957.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym, 2016.

Chen, T., Xu, J., and Agrawal, P. A system for general in-hand object re-orientation. In *Conference on Robot Learning*, pp. 297–307. PMLR, 2022.

Chen, T., Tippur, M., Wu, S., Kumar, V., Adelson, E., and Agrawal, P. Visual dexterity: In-hand reorientation of novel and complex object shapes. *Science Robotics*, 8(84):eadc9244, 2023. doi: 10.1126/scirobotics.adc9244. URL https://www.science.org/doi/abs/10.1126/scirobotics.adc9244.

Chen, X. and He, K. Exploring simple siamese representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 15750–15758, 2021.

Coulom, R. Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games*, pp. 72–83. Springer, 2006.

Danihelka, I., Guez, A., Schrittwieser, J., and Silver, D. Policy improvement by planning with gumbel. In *International Conference on Learning Representations*, 2021.

De Asis, K., Hernandez-Garcia, J., Holland, G., and Sutton, R. Multi-step reinforcement learning: A unifying algorithm. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Feinberg, V., Wan, A., Stoica, I., Jordan, M. I., Gonzalez, J. E., and Levine, S. Model-based value expansion for efficient model-free reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning (ICML 2018)*, 2018.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pp. 1861–1870. PMLR, 2018.

Hafner, D., Lillicrap, T., Ba, J., and Norouzi, M. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019.

Hafner, D., Pasukonis, J., Ba, J., and Lillicrap, T. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023.

Hansen, N., Wang, X., and Su, H. Temporal difference learning for model predictive control. *arXiv preprint arXiv:2203.04955*, 2022.

Hansen, N., Su, H., and Wang, X. Td-mpc2: Scalable, robust world models for continuous control, 2023.

Hubert, T., Schrittwieser, J., Antonoglou, I., Barekatain, M., Schmitt, S., and Silver, D. Learning and planning in complex action spaces. In *International Conference on Machine Learning*, pp. 4476–4486. PMLR, 2021.

Hwangbo, J., Lee, J., Dosovitskiy, A., Bellicoso, D., Tsounis, V., Koltun, V., and Hutter, M. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26):eaau5872, 2019.

Kaiser, L., Babaeizadeh, M., Milos, P., Osinski, B., Campbell, R. H., Czechowski, K., Erhan, D., Finn, C., Kozakowski, P., Levine, S., et al. Model-based reinforcement learning for atari. *arXiv preprint arXiv:1903.00374*, 2019.

Karnin, Z., Koren, T., and Somekh, O. Almost optimal exploration in multi-armed bandits. In *International conference on machine learning*, pp. 1238–1246. PMLR, 2013.

Kool, W., Van Hoof, H., and Welling, M. Stochastic beams and where to find them: The gumbel-top-k trick for sampling sequences without replacement. In *International Conference on Machine Learning*, pp. 3499–3508. PMLR, 2019.

Laskin, M., Srinivas, A., and Abbeel, P. Curl: Contrastive unsupervised representations for reinforcement learning. In *International Conference on Machine Learning*, pp. 5639–5650. PMLR, 2020.

Petrenko, A., Allshire, A., State, G., Handa, A., and Makoviychuk, V. Dexpbt: Scaling up dexterous manipulation for hand-arm systems with population based training. *arXiv preprint arXiv:2305.12127*, 2023.

Rubinstein, R. Y. Optimization of computer simulation models with rare events. *European Journal of Operational Research*, 99(1):89–112, 1997.

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

Schrittwieser, J., Hubert, T., Mandhane, A., Barekatain, M., Antonoglou, I., and Silver, D. Online and offline reinforcement learning by planning with a learned model. *Advances in Neural Information Processing Systems*, 34: 27580–27591, 2021.

Schwarzer, M., Anand, A., Goel, R., Hjelm, R. D., Courville, A., and Bachman, P. Data-efficient reinforcement learning with self-predictive representations. *arXiv preprint arXiv:2007.05929*, 2020.

Schwarzer, M., Ceron, J. S. O., Courville, A., Bellemare, M. G., Agarwal, R., and Castro, P. S. Bigger, better, faster: Human-level atari with human-level efficiency. In *International Conference on Machine Learning*, pp. 30365–30380. PMLR, 2023.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. A general reinforcement learning algorithm

that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

Tassa, Y., Doron, Y., Muldal, A., Erez, T., Li, Y., Casas, D. d. L., Budden, D., Abdolmaleki, A., Merel, J., Lefrancq, A., et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

Wu, P., Escontrela, A., Hafner, D., Goldberg, K., and Abbeel, P. Daydreamer: World models for physical robot learning, 2022.

Xiong, R., Yang, Y., He, D., Zheng, K., Zheng, S., Xing, C., Zhang, H., Lan, Y., Wang, L., and Liu, T. On layer normalization in the transformer architecture. In *International Conference on Machine Learning*, pp. 10524–10533. PMLR, 2020.

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

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

Zhao, D., Tu, S., and Xu, L. Efficient learning for alphazero via path consistency. In *International Conference on Machine Learning*, pp. 26971–26981. PMLR, 2022.

Zheng, H., Yang, Z., Liu, W., Liang, J., and Li, Y. Improving deep neural networks using softplus units. In *2015 International joint conference on neural networks (IJCNN)*, pp. 1–4. IEEE, 2015.

A. Summary of Differences

EfficientZero V2 builds upon EffcientZero algorithm (Ye et al., 2021). This section demonstrates the major applied changes to achieve mastering performance across domains.

- **Search**: Different from the MCTS in EfficientZero, we employ Gumbel search which differs in action selections. Gumbel search guarantees policy improvement even if with limited simulation budgets, which significantly reduces the computation of EZ-V2.
- **Search-based Value Estimation**: Compared to the adaptive TD method used in EfficientZero's value estimation, we employ the empirical mean of the search root node as the target value. The search process utilizes the current model and policy to calculate improved policy and value estimation, which can improve the utilization of early-stage transitions.
- **Gaussian Policy**: Inspired by Sampled MuZero (Hubert et al., 2021), we employ an Gaussian distribution, which is parameterized by the learnable policy function, to represent the policy in continuous action spaces. We generate search action candidates by simply sampling from the Gaussian policy, which naturally satisfies the sampling without replacement in Gumbel search. We then prove the policy improvement of Gumbel search still holds in the continuous setting.
- **Action Embedding**: We employ an action embedding layer to encode the real actions as latent vectors. By representing actions in a hidden space, actions that are similar to each other are placed closer in the embedding space. This proximity allows the RL agent to generalize its learning from one action to similar actions, improving its efficiency and performance.
- **Priority Precalculation**: Conventionally, the priorities of a newly collected trajectory are set to be the maximal priority of total collected transitions. We propose to warm up the new priorities by calculating the bellman error using the current model. This increases the probability of newly collected transitions being replayed, thereby improving sample efficiency.
- **Architecture**: For the 2-dimensional image inputs, we follow the most implementation of the architecture of EfficientZero. For the continuous control task with 1-dimensional input, we use a variation of the previous architecture in which all convolutions are replaced by fully connected layers. The details could be found in Appendix G.
- **Hyperparameters**: We tuned the hyperparameters of EfficientZero V2 to achieve satisfying performance across various domains. The generality is verified by training from scratch without further adjustments, including Atari 100k, Proprio Control, Vision Control. More details refer to Appendix I.

B. Calculation of Target Policy

In discrete control, the calculation of the target policy is the same as that of the original Gumbel search, which can be found in Gumbel Muzero (Danihelka et al., 2021). In a continuous setting, we modify the calculation of target policy as follows.

$$\pi' = \text{softmax}(\sigma(\text{completedQ})) \tag{18}$$

where completedQ is a comprehensive value estimation of visited and unvisited candidates. For visited nodes, the completedQ is calculated via the empirical estimation as $q(a) = r(s,a) + \gamma v(s')$, $v(s')$ is the empirical mean of bootstrapped value sum and visit counts. For non-visits, completedQ is estimated through the weighted average Q-value of visited nodes as $\sum_a \pi(a) q(a)$.

We mention the transformation σ is a monotonically increasing transformation in the main text. For a concrete instantiation of σ, we use

$$\sigma(q(s,a)) = \left(c_{\text{visit}} + \max_b N(b)\right) c_{\text{scale}}\, q(s,a) \tag{19}$$

where $\max_b N(b)$ is the visit count of the most visited action. c_{visit} and c_{scale} are 50 and 0.1 respectively.

C. Advantage of Simple Policy Loss

Different from minimizing the cross-entropy between the policy with the target policy, the simple loss using a_S^* directly improves the possibility of the recommended action in tree search. That means we can make the policy network output a good action in the early stage. With the iteration repeating, the policy network can find the optimal point. More specifically, we provide an intuitive example to illustrate that the policy can reach the optimal point more efficiently, as shown in Fig. 5. The whole space represents the action space. The simple loss enforces the policy to output the current best point colored red. The original cross-entropy loss considers all actions and thus makes the policy's output near the point colored brown. We can see that when the action space is large, the red point can guide the policy to reach the optimal point colored purple more quickly.

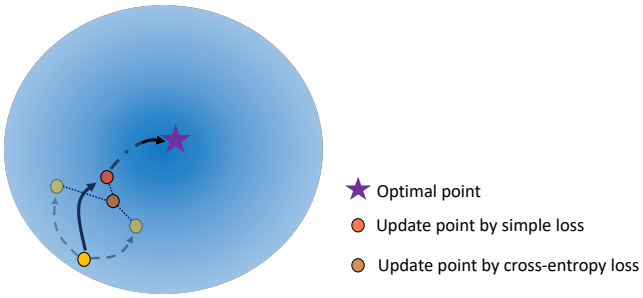

Figure 5. Intuitive example showing the difference between simple policy loss using a_S^* and cross-entropy loss.

D. Details of Search-based Value Estimation (SVE)

D.1. Calculation of SVE

First, the search process expands a tree through N simulations gradually. At each simulation, the agent dives to a leaf and expands a new child. The diving path is easily associated with an $H(n)$-step rollout, which forms an imagined H(n)-step value estimation for the root node. This estimation process will be repeated N times, to get an average estimation, as defined in Definition D.1.

Definition D.1 (Search-Based Value Estimation). *Using imagined states and rewards $\hat{s}_{t+1}, \hat{r}_t = \mathcal{G}(\hat{s}_t, \hat{a}_t)$ obtained from our learnable dynamic function, the value estimation of a given state s_0 can be derived from the empirical mean of N bootstrapped estimations, which is formulated as*

$$\hat{V}_{\mathrm{S}}(s_0) = \frac{\sum_{n=0}^{N} \hat{V}_n(s_0)}{N} \tag{20}$$

where N denotes the number of simulations, $\hat{V}_n(s_0)$ is the bootstrapped estimation of the n-th node expansion, which is formulated as

$$\hat{V}_n(s_0) = \sum_{t=0}^{H(n)} \gamma^t \hat{r}_t + \gamma^{H(n)} \hat{V}(\hat{s}_{H(n)}) \tag{21}$$

where $H(n)$ denotes the search depth of the n-th iteration.

E. Proof for SVE Error Bound

Corollary E.1 (Search-Based Value Estimation Error). *Define s_t, a_t, r_t to be the states, actions, and rewards resulting from current policy π using true dynamics $\mathcal{G}^*$ and reward function $\mathcal{R}^*$, starting from $s_0 \sim \nu$ and similarly define $\hat{s}_t, \hat{a}_t, \hat{r}_t$ using learned function $\mathcal{G}$. Let reward function $\mathcal{R}$ to be $L_r - Lipschitz$ and value function $\mathcal{V}$ as $L_V - Lipschitz$. Assume $\epsilon_s, \epsilon_r, \epsilon_v$ as upper bounds of state transition, reward, and value estimations respectively. We define the error bounds of each*

estimation as

$$\max_{n\in[N],t\in[H(n)]} \mathbb{E}\left[\|\hat{s}_t - s_t\|^2\right] \leq \epsilon_s^2 \tag{22}$$

$$\max_{n\in[N],t\in[H(n)]} \mathbb{E}\left[\|\mathcal{R}(s_t) - \mathcal{R}^*(s_t)\|^2\right] \leq \epsilon_r^2 \tag{23}$$

$$\max_{n\in[N],t\in[H(n)]} \mathbb{E}\left[\|\mathcal{V}(s_t) - \mathcal{V}^*(s_t)\|^2\right] \leq \epsilon_v^2 \tag{24}$$

within a tree-search process. Then we have errors

$$\textit{MSE}_\nu(\hat{V}_S) \leq \frac{4}{N^2}\sum_{n=0}^{N}\left(\sum_{t=0}^{H(n)}\gamma^{2t}(L_r^2\epsilon_s^2+\epsilon_r^2)+\gamma^{2H(n)}(L_V^2\epsilon_s^2+\epsilon_v^2)\right) \tag{25}$$

where N is the simulation number of the search process. $H(n)$ denotes the depth of the n-th search iteration.

Proof. To provide detailed proof for the upper bound of the MSE of the search-based value estimation $\hat{V}_{\text{S}}$, we follow a structured approach inspired by Model-based Value Estimation (MVE) (Feinberg et al., 2018), with adjustments for the specifics of MCTS and considerations of errors of reward and value estimations.

Given Definition 4.2, we aim to bound the MSE of this estimator, defined as:

$$\text{MSE}_\nu(\hat{V}_{\text{S}}) = \mathbb{E}\left[\left(\hat{V}_{\text{S}}(s_0) - V_\pi(s_0)\right)^2\right] \tag{26}$$

Where $V_\pi(s_0) = \sum_{t=0}^{H-1}\gamma^t r_t + \gamma^H V_\pi(s_H)$. We can first decompose the MSE as

$$\text{MSE}_\nu(\hat{V}_{\text{S}}) = \mathbb{E}\left[\left(\frac{1}{N}\sum_{n=0}^{N}\left(\sum_{t=0}^{H(n)}\gamma^t(\hat{r}_t - r_t) + \gamma^{H(n)}\left(\hat{V}(\hat{s}_{H(n)}) - V_\pi(s_{H(n)})\right)\right)\right)^2\right] \tag{27}$$

According to $L-Lipschitz$ continuity of $\mathcal{R}$ and Cauchy inequality, we can derive

$$\begin{aligned}
\mathbb{E}\left[(\hat{r}_t - r_t)^2\right] &= \mathbb{E}\left[(\mathcal{R}(\hat{s}_t) - \mathcal{R}^*(s_t))^2\right] \\
&= \mathbb{E}\left[(\mathcal{R}(\hat{s}_t) - \mathcal{R}(s_t) + \mathcal{R}(s_t) - \mathcal{R}^*(s_t))^2\right] \\
&\leq 2\mathbb{E}\left[(\mathcal{R}(\hat{s}_t) - \mathcal{R}(s_t))^2\right] + 2\mathbb{E}\left[(\mathcal{R}(s_t) - \mathcal{R}^*(s_t))^2\right] \text{ (Cauchy inequality)} \\
&\leq 2L_r^2\mathbb{E}\left[\|\hat{s}_t - s_t\|^2\right] + 2\mathbb{E}\left[\|\mathcal{R}(s_t) - \mathcal{R}^*(s_t)\|^2\right] \\
&\leq 2L_r^2\epsilon_s^2 + 2\epsilon_r^2
\end{aligned}$$

Similarly, we derive the error bound of value estimation as

$$\mathbb{E}\left[\left(\hat{V}(\hat{s}_t) - V_\pi(s_t)\right)^2\right] = \mathbb{E}\left[(\mathcal{V}(\hat{s}_t) - \mathcal{V}^*(s_t))^2\right] \leq 2L_v^2\epsilon_s^2 + 2\epsilon_v^2$$

Assume the model inference errors are additive per step, hence

$$\begin{aligned}
\text{MSE}_\nu(\hat{V}_{\text{S}}) \leq& \frac{2}{N^2}\sum_{n=0}^{N}\left(\sum_{t=0}^{H(n)}\gamma^{2t}(\hat{r}_t - r_t)^2 + \gamma^{2H(n)}\left(\hat{V}(\hat{s}_{H(n)}) - V_\pi(s_{H(n)})\right)^2\right) \\
\leq& \frac{4}{N^2}\sum_{n=0}^{N}\left(\sum_{t=0}^{H(n)}\gamma^{2t}(L_r^2\epsilon_s^2+\epsilon_r^2)+\gamma^{2H(n)}(L_V^2\epsilon_s^2+\epsilon_v^2)\right)
\end{aligned} \tag{28}$$

Considering the convergence with increasing search depth $H(n)$, we separate it into two parts:

1. **The reward error term** $\sum_{t=0}^{H(n)} \gamma^{2t}(L_r^2\epsilon_s^2 + \epsilon_r^2)$: Given $L_r \in \mathcal{C}$, $\gamma \in (0, 1)$, and ϵ_s, ϵ_r decaying with training, the series is obviously convergent.

2. **The terminal state value error term** $\gamma^{2H(n)}(L_V^2\epsilon_s^2 + \epsilon_v^2)$ is similarly converged.

On the other hand, this upper bound will also converge to 0 if the model is optimal $\epsilon_s, \epsilon_r, \epsilon_v \to 0$. □

Compared to SVE, the estimation error of multi-step TD methods in (De Asis et al., 2018) depends on sampling, making it difficult to determine their theoretical upper bound of estimation errors. In other words, the estimation error of multi-step TD methods increases gradually during learning due to the off-policy issue.

F. Details of Mixed Value Target

The mixed value target is calculated for each transition sampled from the buffer. It contains two types of value targets: the search-based value target and the multi-step TD target.

Specifically, we use two criteria to determine if we should use a search-based value target. The first criterion is whether the sampled transition comes from recent rollouts. If the transition is not from recent rollouts, meaning the transition is considered stale, we opt for the search-based value as the target. Otherwise, the multi-step TD (Temporal Difference) target is employed. The second criterion considers whether the current training step is in the early stage. During this phase, all transitions in the buffer are fresh. Additionally, the error in the dynamic model is still significant, resulting in inaccuracies in search-based value estimation. Therefore, we opt for the multi-step TD target as the target value.

G. Details of Achitecture

For the 2-dimensional image inputs, we follow the most implementation of the architecture of EfficientZero. For the continuous control task with 1-dimensional input, we use a variation of the previous architecture in which all convolutions are replaced by fully connected layers. In the following, we describe the detailed architecture of EZ-V2 under 1-dimensional input.

The representation function $\mathcal{H}$ first processes the observation by a running mean block. The running mean block is similar to a Batch Normalization layer without learnable parameters. Then, the normalized input is processed by a linear layer, followed by a Layer Normalisation and a Tanh activation. Hereafter, we use a Pre-LN Transformer style pre-activation residual tower (Xiong et al., 2020) coupled with Layer Normalisation and Rectified Linear Unit (ReLU) activations to obtain the latent state. We used 3 blocks and the output dim is 128. Each linear layer has a hidden size of 256.

The dynamic function $\mathcal{G}$ takes the state and the action embedding as inputs. the action embedding is obtained from the action embedding layer which consists of a linear layer, a Layer Normalization, and a ReLU activation. The size of the action embedding is 64. The combination of the state and the action embedding are also processed by Pre-LN Transformer style pre-activation residual tower (Xiong et al., 2020) coupled with Layer Normalisation and ReLU activations.

For the reward $\mathcal{R}$, value $\mathcal{V}$ and policy $\mathcal{P}$ function share the similar network structures. Taking the state as input, a linear layer followed by a Layer Normalization obtains the hidden variables. Then, we use the MLP network combined with Batch Normalization, which is similar to that of EfficientZero, to obtain the reward, value, and policy prediction. The hidden size of each layer is 256, and the activation function is ReLU.

The reward and value predictions used the categorical representation introduced in EfficientZero. We used 51 bins for both the value and the reward predictions with the value being able to represent values between $\{-299.0, 299.0\}$. The reward can represent values between $\{-2.0, 2.0\}$. The maximum reward is 2 because the action repeat in DMControl is 2. For policy function, the network outputs the mean and standard deviation of the Gaussian distribution. Then we use the 5 times Tanh function to restrict the range of the mean, and Softplus (Zheng et al., 2015) function to make the standard deviation over 0. In addition, the policy distribution is modeled by a squashed Gaussian distribution. A squashed Gaussian distribution belongs to a modification of a standard Gaussian, where the outputs are transformed into a bounded interval.

Furthermore, we add a running mean block for observation in the representation function $\mathcal{H}$ in continuous control whose observation is 1 dimension. A key benefit of the module is that it normalizes the observation to mitigate exploding gradients.

H. Training pipeline

The training pipeline comprises data workers, batch workers, and a learner. The data workers, also known as self-play workers, collect trajectories based on the model updated at specific intervals. The actions executed in these trajectories are determined using the sampling-based Gumbel search, as depicted in Figure 2 (B).

On the other hand, batch workers provide batch transitions sampled from the replay buffer to the learner. Similar to EfficientZero, the target policy and value in batch transitions are reanalyzed with the latest target model. This reanalysis involves revisiting past trajectories and re-executing the data using the target model, resulting in fresher search-based values and target policies obtained through model inference and Gumbel search. The target model undergoes periodic updates at specified intervals during the training process.

Finally, the learner trains the reward, dynamics, value, and policy functions using the reanalyzed batch. Figure 2 (A) and Equation (5) illustrate the specific losses involved in the training process. To enhance the learning process, we have designed a parallel training framework where data workers, batch workers, and a learner all operate concurrently.

I. Hyperparameters of Algorithms

I.1. Hyperparameters of Our Method

We employ similar hyperparameters across all domains, as outlined in Table 3. It's worth noting that we use different optimizers in tasks with different inputs. Specifically, due to architectural differences, we opt for the Adam optimizer in the 'Proprio Control 50-100k' task, whereas we utilize the SGD optimizer in 'Vision Control 100-200k' and 'Atari 100k'.

In the case of most baselines, we either adhere to the suggested hyperparameters provided by the authors of those baselines for each domain or fine-tune them to suit our setup when such suggestions are not available. Notably, SAC, DrQ-v2, and DreamerV3 employ a larger batch size of 512, while our method and EfficientZero achieve stable learning with a batch size of 256.

I.2. Hyperparameters of Baselines

I.2.1. DreamerV3

We use the official reimplementation of DreamerV3, which can be found at https://github.com/danijar/dreamerv3. In line with the original authors' recommendations, the results we used are based on their suggested hyperparameters and the S model size for Atari 100K, Proprio Control 50-100k, and Vision Control 100-200k. For a comprehensive list of hyperparameters, please refer to their published paper (Hafner et al., 2023).

I.2.2. TD-MPC2

We benchmark against the official implementation of TD-MPC2 available at https://github.com/nicklashansen/tdmpc2. We reproduce results according to the official implementation, which is shown in Fig. 6. We use the suggested hyperparameters and select the default 5M trainable parameters. Refer to their paper (Hansen et al., 2023) for a complete list of hyperparameters.

I.2.3. SAC

We follow the SAC implementation from https://github.com/ denisyarats/pytorch_sac, and we use the hyperparameters suggested by the authors (when available). Refer to their repo for a complete list of hyperparameters.

I.2.4. DrQ-v2

We follow the DrQ-v2 implementation from https://github.com/facebookresearch/drqv2, and we use the hyperparameters suggested by the authors (when available). Refer to their repo for a complete list of hyperparameters.

I.2.5. EfficientZero

We conduct benchmarks using the official reimplementation of EfficientZero, which can be found at https://github.com/YeWR/EfficientZero. In line with the original authors' recommendations, we used their suggested

Name	Symbol	Value
General		
Replay capacity (FIFO)	—	10^6
Batch size	$\mathcal{B}$	256
Discount	γ	0.997
Update-to-data (UTD)	—	1
Unroll steps	l_{unroll}	5
TD steps	k	5
Number of simulations in search	N_{sim}	32 (16 in 'Atari 100k')
Number of sampled actions	K	16 (8 in 'Atari 100k')
Self-play network updating interval	—	100
Target network updating interval	—	400
Starting steps when using SVE	T_1	$4 \cdot 10^4$
Threshold of buffer index when using SVE	T_2	$2 \cdot 10^4$
Priority exponent	α	1
Priority correction	β	1
Reward loss coefficient	λ_1	1.0
Policy loss coefficient	λ_2	1.0
Value loss coefficient	λ_3	0.25
Self-supervised consistency loss coefficient	λ_4	2.0
Policy entropy loss coefficient	—	$5 \cdot 10^{-3}$
Proprio Control 50-100k		
Optimizer	—	Adam
Optimizer: learning rate	—	$3 \cdot 10^{-4}$
Optimizer: weight decay	—	$2 \cdot 10^{-5}$
Vision Control 100-200k & Atari 100k		
Optimizer	—	SGD
Optimizer: learning rate	—	0.2
Optimizer: weight decay	—	$1 \cdot 10^{-4}$
Optimizer: momentum	—	0.9

Table 3. hyper-parameters of EfficientZero V2.

hyperparameters for Atari 100K. For a comprehensive list of hyperparameters, please refer to their published paper (Ye et al., 2021).

I.2.6. BBF

We use the official results of BBF, which can be found at https://github.com/google-research/google-research/tree/master/bigger_better_faster.

J. Details of Experiments

J.1. Comparison

We present the training curves across various benchmarks, including Atari 100k, Proprio Control 50-100k, and Vision Control 100-200k. Atari 100k, featuring 26 games, is a widely used benchmark for assessing the sample efficiency of different algorithms. In the case of Proprio Control 50-100k and Vision Control 100-200k, we have considered 20 continuous control tasks for each. You can find the training curves for EZ-V2 and the baselines in Figures 6, 7, and 8.

J.2. Ablation

Additionally, we include an ablation study focusing on the sampling-based Gumbel search and the mixed value target. Table 4, 5 and 6 demonstrate that our search method and mixed value target achieve superior performance on tasks with proprioceptive and image inputs. The action space is 1-dimensional in tasks such as Acrobot Swingup, Cartpole Swingup Sparse, and Pendulum Swingup. The dimension is greater than 2 in other DM-Control tasks. For Atari 100k, we selected 3 of the relatively most challenging tasks for our ablation study.

We have observed that our mixed value target outperforms other value estimation methods in most tasks, such as multi-step value target and generalized advantage estimation (GAE). This indicates that the method mitigating the off-policy issue consistently performs better. Compared to Sample MCTS, the S-Gumbel Search method significantly enhances performance in tasks with a high-dimensional action space. The results demonstrate that the S-Gumbel Search method can achieve superior performance with the limited simulations.

Table 4. Proprio Control 50-100k

Method	Acrobot swingup	Cartpole swingup sparse	Pendulum swingup	Reacher hard	Walker walk	Walker run	Quadruped walk
Original EZ-v2	**297.7**	**795.4**	825.4	**795.4**	**944.0**	**657.2**	**925.8**
W/ Multi-step TD Target	256.7	473.6	**836**	590.7	839.3	521.6	649.7
W/ GAE	275.1	766.7	336	601.6	757.3	512.7	719.7
W/ Sample MCTS	248.1	789.3	357.4	754.7	691.7	381.1	254.7

Table 5. Vision Control 100-200k

Method	Acrobot swingup	Cartpole swingup sparse	Pendulum swingup	Reacher hard	Walker walk	Walker run	Quadruped walk
Original EZ-v2	**231.8**	763.6	**726.7**	**961.5**	**888.8**	475.3	**433.3**
W/ Multi-step TD Target	186.3	631.7	411.4	878	880.5	**496.7**	410.0
W/ GAE	122.0	**796.1**	718.0	934.9	765.1	491.5	299.3
W/ Sample MCTS	73.9	786.1	372.9	938.8	619.9	264.2	141.4

Table 6. Atari 100k

	Game		
Method	Asterix	UpNDown	Qbert
Original EZ-v2	**61810.0**	15224.3	**16058.3**
W/ Multi-step TD Target	26023.1	13725.7	9463.3
W/ GAE	22816.7	**16255.3**	7807.3

J.3. Computation Load

We have included practical comparisons between the TD-MPC2 and EZ-V2 algorithms in terms of computational load, using the 'Walker Run' task as an example. The following results include the parameter count, FLOPs (per decision step) and training time for each algorithm. The methods were benchmarked on a server equipped with 8 RTX 3090 graphics cards.

EZ-V2 requires 1000 times fewer FLOPs per decision step compared to TD-MPC2, while also using almost 4 times fewer parameters. The decision step, crucial for collecting interaction data, occurs during the evaluation. This remarkable efficiency arises from two primary factors:

- The planning process in TD-MPC2, utilizing the MPPI method, involves predicting 9216 latent states. In contrast, our method extends only 32 latent states, significantly reducing the computational load.
- TD-MPC2 relies on an ensemble of Q-functions (5 heads), and its latent state dimension is 4 times larger than our method's, leading to higher computational demands.

The FLOPs per decision step are vital for deployments, especially in robotics control. Since robots typically have limited edge computing resources, using less computation is essential for real-time tasks. TD-MPC2 needs more computational

Table 7. Model Parameters and Training Performance

	Parameters	Flops per decision step	Time per 100k training (h)
EZ-V2	$\mathbf{1.3 \times 10^6}$	$\mathbf{4.7 \times 10^7}$	**2.7**
TD-MPC2	4.9×10^6	3.6×10^{10}	3.3

resources and high-performance computing. In contrast, our method runs efficiently with less demand of computing resources, making it ideal for real-time robotic planning.

Regarding training time, both methods require a similar amount of time per 100k training steps. This similarity in time consumption is due to that EZ-V2 and TD-MPC2 share similar unrolled training frameworks with the same batch size. EZ-V2 wins in speed due to its distributed implementation, without rollout and evaluation overheads in the training process.

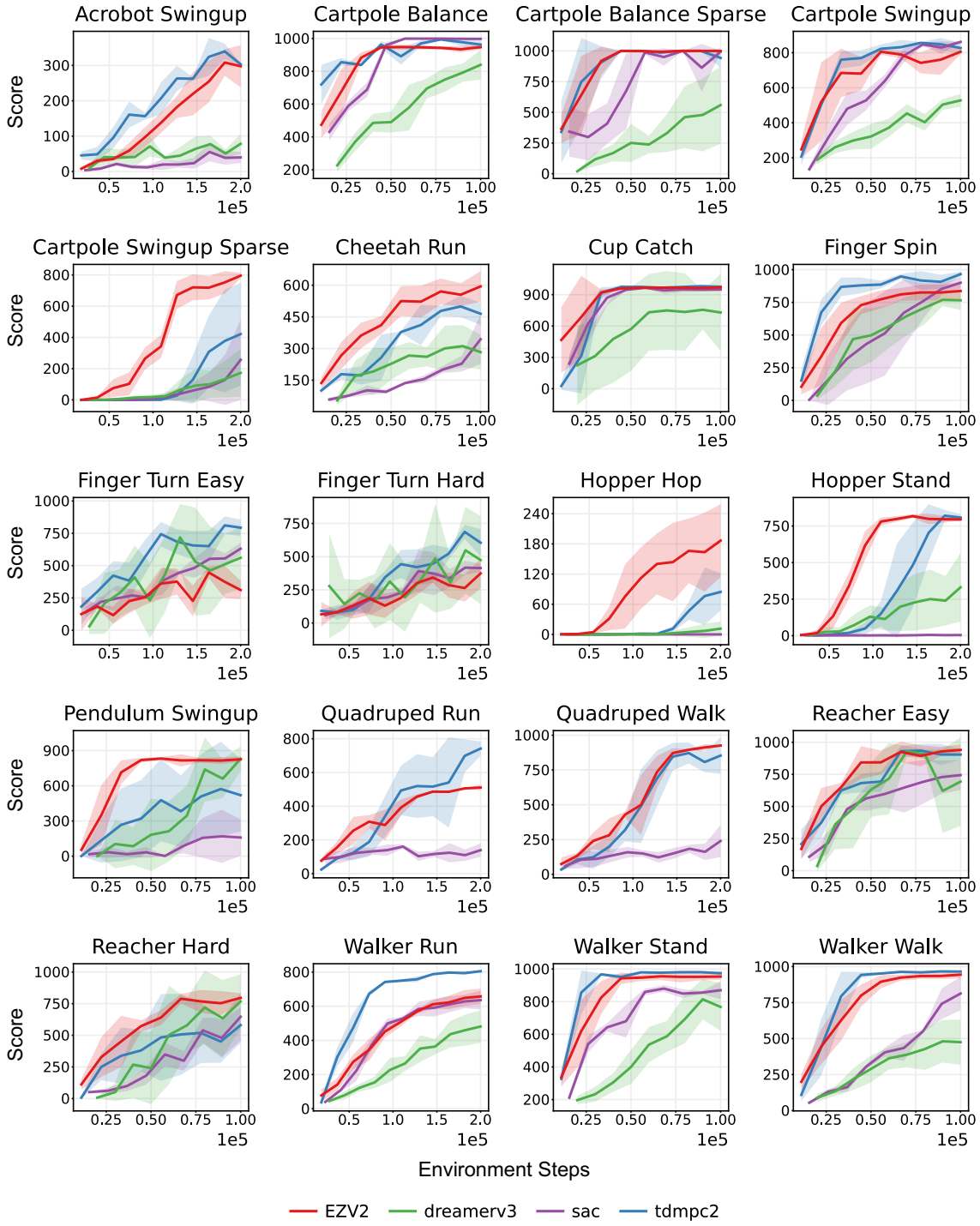

Figure 6. DMC scores for proprioceptive inputs with a budget of 200K frames. it corresponds to 100K steps due to the action repeat. (Because different algorithms have varying logging frequencies, the starting points are not the same.)

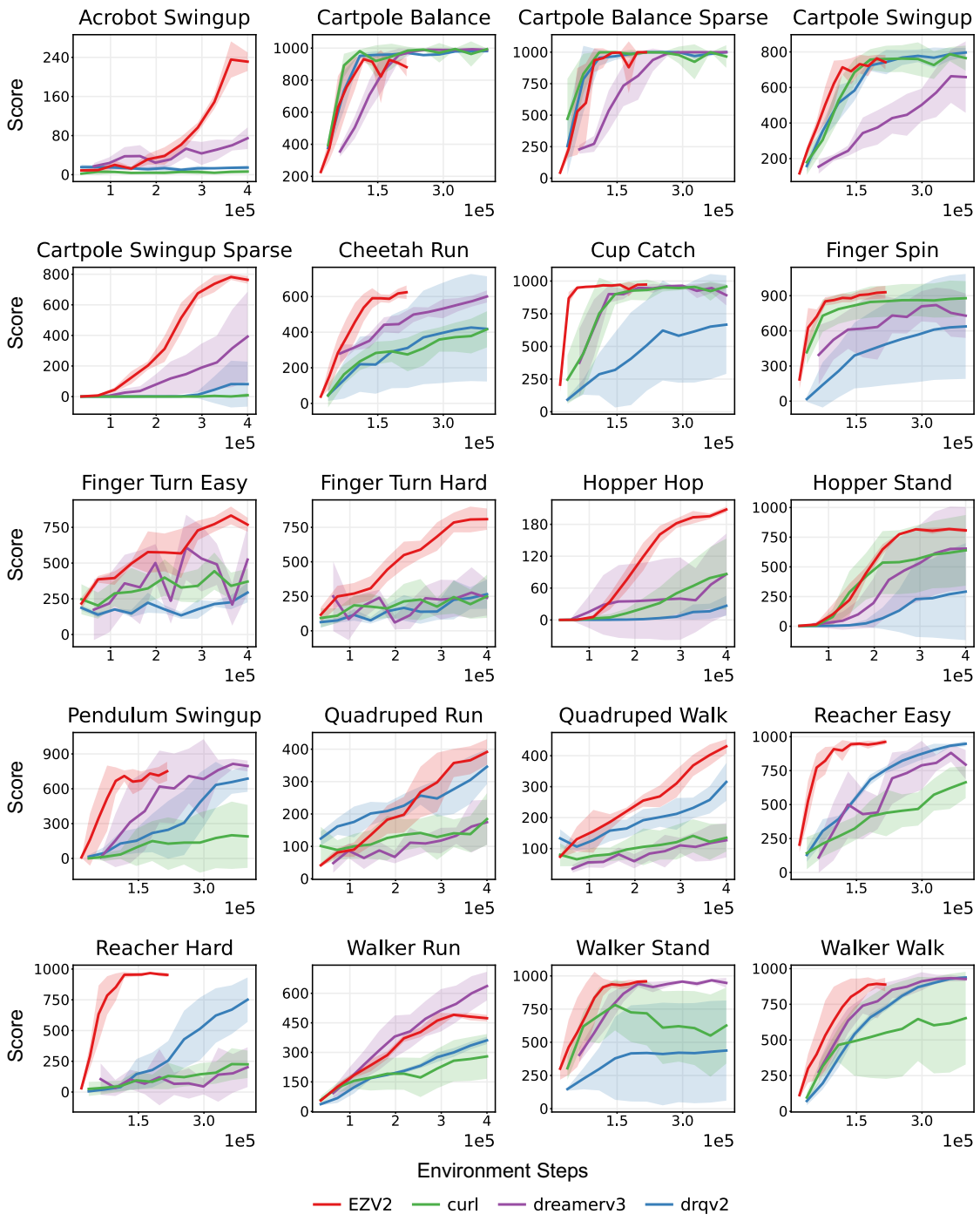

Figure 7. DMC scores for image inputs with a budget of 400K frames. It corresponds to 200K steps due to the action repeat. (Because different algorithms have varying logging frequencies, the starting points are not the same.)

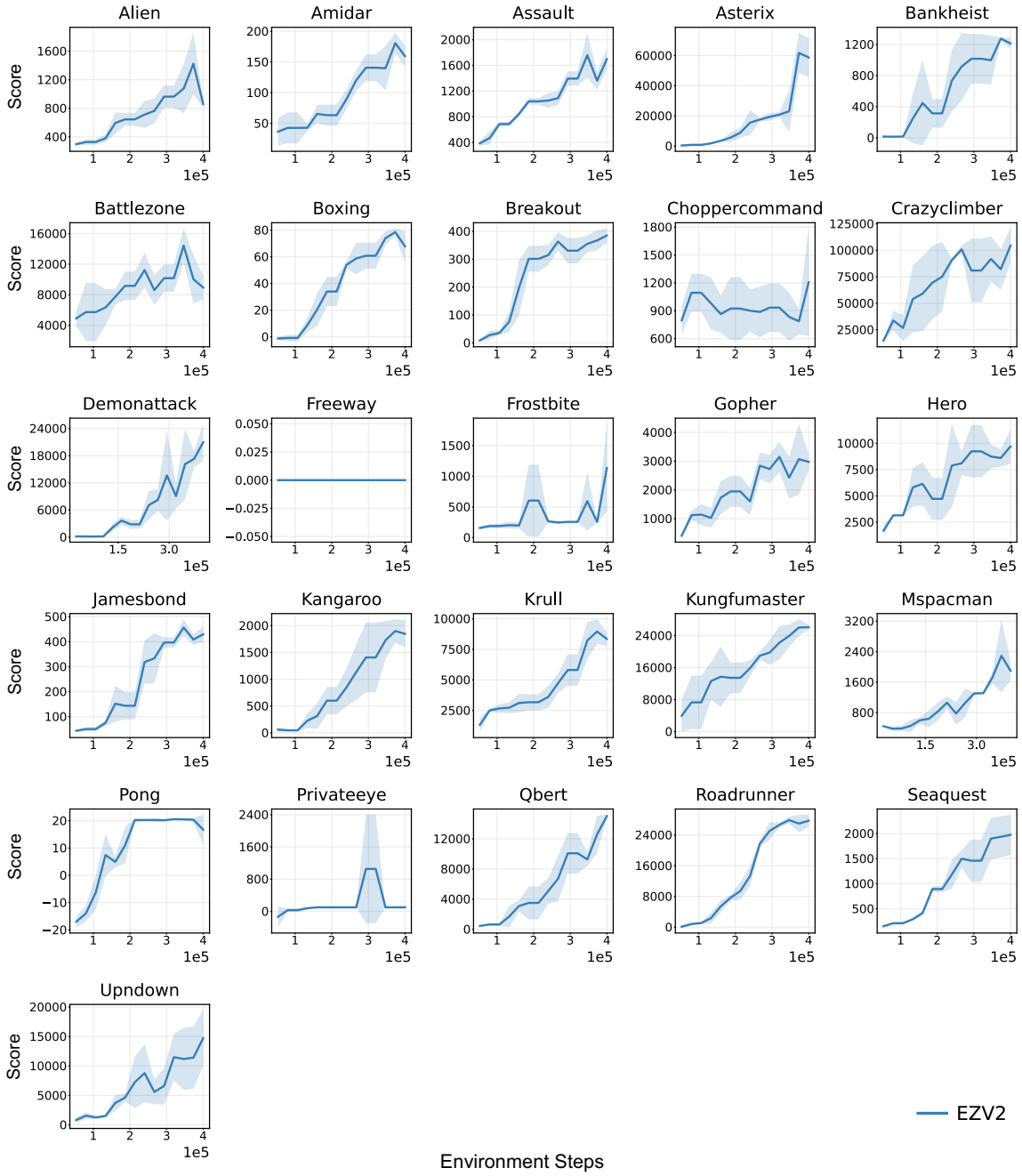

Figure 8. Atari training curves with a budget of 400K frames, amounting to 100K interaction data.