ReZero: Boosting MCTS-based Algorithms by Just-in-Time and Speedy Reanalyze

Chunyu Xuan [1 2] **Yazhe Niu** [2 3] **Yuan Pu** [2] **Shuai Hu** [3] **Yu Liu** [2 3] **Jing Yang** [1]

Abstract

MCTS-based algorithms, such as MuZero and its derivatives, have achieved widespread success in various decision-making domains. These algorithms employ the *reanalyze* process to enhance sample efficiency, albeit at the expense of significant wall-clock time consumption. To address this issue, we propose a general approach named ReZero to boost MCTS-based algorithms. Specifically, we propose a new scheme that simplifies data collecting and reanalyzing, which significantly reduces the search cost while guarantees the performance as well. Furthermore, to accelerate each search process, we conceive a method to reuse the subsequent information in the trajectory. The corresponding analysis conducted on the bandit model also provides auxiliary theoretical substantiation for our design. Experiments conducted on Atari environments and board games demonstrates that ReZero substantially improves training speed while maintaining high sample efficiency. The code is available as part of the LightZero benchmark at `https://github.com/opendilab/LightZero`.

1. Introduction

As a pivotal subset of artificial intelligence, Reinforcement Learning (RL) (Sutton & Barto, 1988) has acquired achievements and applications across diverse fields, including interactive gaming (Vinyals et al., 2019), autonomous vehicles (Li et al., 2022), and natural language processing (Rafailov et al., 2023). Despite its successes, a fundamental challenge plaguing traditional model-free RL algorithms is their low sample efficiency. These algorithms typically require a large amount of data to learn effectively, making them often infeasible in the real world. In response to this problem, numerous model-based RL(Janner et al., 2019) (Hafner et al., 2020) methods have emerged. These methods involve learning an additional model of the environment from data and utilizing the model to assist the agent's learning, thereby improving sample efficiency significantly.

Within this field, Monte Carlo Tree Search (Świechowski et al., 2023) (MCTS) has been proven to be a efficient method for utilizing models for planning. It incorporates the UCB1 algorithm (Auer et al., 2002) into the tree search process and has achieved promising results in in a wide range of application scenarios. Specifically, AlphaZero (Silver et al., 2017) plays a big role in combining deep RL with MCTS, achieving notable accomplishments that can beat top-level human players. While AlphaZero can only be applied to environments with perfect simulators, MuZero (Schrittwieser et al., 2019) extended the algorithm to cases without known environment models, resulting in good performances in a wider range of tasks. Following MuZero, many successor algorithms have emerged, enabling MuZero to be applied in continuous action spaces (Hubert et al., 2021), environments with stochastic dynamics (Antonoglou et al., 2021), offline RL training scenarios (Schrittwieser et al., 2021), and etc. All these MCTS-based algorithms (Silver et al., 2016; 2017; Schrittwieser et al., 2019) made valuable contributions to the universal applicability of the MCTS+RL framework.

However, the extensive tree search computations incur additional time overhead for these algorithms: During the data collection phase, the agent needs to execute MCTS to select an action every time it receives a new state. Furthermore, due to the characteristics of tree search, it is challenging to parallelize it using commonly used vectorized environments methods, further amplifying the speed disadvantage. On the other hand, during the reanalyze (Schrittwieser et al., 2021) process, in order to obtain higher-quality update targets, the latest model parameters are used to re-run MCTS on samples in buffer. The time required to process samples thus increases as a trade-off for high sample efficiency. The excessive time cost has become a bottleneck hindering the further promotion of these algorithms.

Recently, a segment of research endeavors are directed toward mitigating the above wall-clock time overhead about MCTS-based algorithms. On the one hand, SpeedyZero (Mei et al., 2023) greatly diminishes algorithms' time over-

[1]Xi'an Jiaotong University, Xi'an, China [2]Shanghai Artificial Intelligence Laboratory, Shanghai, China [3]SenseTime Research, Shanghai, China. Correspondence to: Jing Yang <jasmine1976@xjtu.edu.cn>.

heads by deploying a parallel system; however, it demands more computational resources. On the other hand, it remains imperative to identify methodologies that accelerate these algorithms without imposing extra computational demands. PTSAZero (Fu et al., 2023), for instance, compresses the search space via state abstraction, decreasing the time cost per search. Nevertheless, it is widely acknowledged that MCTS-based algorithms necessitate search times to assure their performance. From this perspective, our research try to discover strategies that preserve algorithmic performance while minimizing the search times as much as possible. Moreover, given the observation that the tree search trajectories for states at adjacent time-steps frequently exhibit a significant degree of overlap, it becomes feasible to leverage the reuse of search information across different time-steps as a method to further expedite each search process.

In this paper, we introduce a new approach designed to boost the MCTS-based algorithms. Firstly, drawing inspiration from the utilization of a periodically updated target network in DQN (Mnih et al., 2013), we find that it is not essential to reanalyze the mini-batch at each iteration. Therefore, we propose a simplified framework which periodically reanalyzes the buffer to update policy targets in a just-in-time manner. Such an approach markedly diminishes the number of calls to MCTS. Secondly, within the reanalyze process, given the collected game trajectory, we propose to reuse search information from subsequent state to accelerate the search of the current state. In addition, we provide theoretical analysis on the potential implications of these modifications about the quality of target policy. Our approach can be easily applied to the MCTS-based algorithm family and can be synergistically implemented alongside existing acceleration methods. In addition, it will not bring about an increase in hardware overhead. Empirical experiments demonstrate that our approach yields good results in both single-agent environment (Atari (Bellemare et al., 2013)) and two-player board games (Silver et al., 2016), greatly improving the training speed while maintaining high sample efficiency. Ablation experiments explore the impact of different reanalyze periods and the acceleration effect of information reuse on a single search. The main contributions of this paper can be summarized as follows:

- We propose a simplified framework that greatly reduces the number of calls to MCTS, thus speeding up MCTS-based algorithms.
- We propose a method to speed up a single search by reusing search information. Theoretical support for the effectiveness of the method is also provided.
- We conduct serveal experiments on diverse environments and further investigate our method through ablations.

2. Related Work

2.1. MCTS-based Algorithms

AlphaGo (Silver et al., 2016) and AlphaZero (Silver et al., 2017) combined MCTS with deep RL, achieving significant results in board games, defeating world-champion players, and revealing their remarkable capabilities. Following these achievements, MuZero (Schrittwieser et al., 2019) combines tree search with a learned value equivalent model (Grimm et al., 2020b), successfully promoting the algorithm's application to scenarios without known models, such as Atari. Subsequent to MuZero's development, several new works based on the MuZero framework emerged. Ye et al. (2021) further improved MuZero's sample efficiency, using very little data for training and still achieving outstanding performance. Hubert et al. (2021) and Antonoglou et al. (2021) extended the MuZero agent to environments with large action spaces and stochastic environments, further broadening the algorithm's usage scenario. Schrittwieser et al. (2021) applied MuZero's reanalyze operation as a policy enhancement operator and extended it to offline settings. Danihelka et al. (2022) changed the node selection method during tree search, so the policy obtained from the search shows a monotonic improvement, ensuring algorithm performance even with low simulation numbers. Despite the aforementioned excellent improvement techniques, the current time overhead of MCTS-based RL remains significant, which is the main problem this paper aims to address.

2.2. MCTS Acceleration

Recent research has focused on accelerating MCTS-based algorithms. The work by Mei et al. (2023) significantly reduces the algorithm's time overhead by designing a parallel system; however, this approach requires more computational resources and involves some adjustments for large batch training. Fu et al. (2023) narrows the search space through state abstraction, which amalgamates redundant *spatial* information, thereby reducing the time cost per search. Silver et al. (2016); Wu (2019) use information reuse to speed up the search process. They save sub-trees of the search tree and serve as initialization for the next search. However, this naive front-to-back information reuse is significantly different from our approach. To our knowledge, we are the first to enhance MCTS by reusing information in a backwards manner. Our proposed approach can seamlessly integrate with various MCTS-based algorithms, many of which (Danihelka et al., 2022; Mei et al., 2023; Fu et al., 2023; Leurent & Maillard, 2020) are orthogonal to our contributions.

3. Preliminaries

3.1. MuZero

MuZero (Schrittwieser et al., 2019) is a fundamental model-based RL algorithm that incorporates a value-equivalent model (Grimm et al., 2020a) and leverages Monte Carlo Tree Search for planning within a learned latent space. The model consists of three core components: a *representation* model h_θ, a *dynamics* model g_θ, and a *prediction* model f_θ:

$$\begin{array}{ll} \text{Representation:} & s_t = h_\theta(o_{t-l:t}) \\ \text{Dynamics:} & s_{t+1}, r_t = g_\theta(s_t, a_t) \\ \text{Prediction:} & v_t, p_t = f_\theta(s_t) \end{array} \tag{1}$$

During the *inference* phase, the representation model transforms a sequence of the last l observations $o_{t-l:t}$ into a corresponding latent state representation s_t. The dynamics model processes this latent state alongside an action a_t, yielding the subsequent latent state s_{t+1} and an estimated reward r_t. Finally, the prediction model accepts a latent state and produces both the predicted policy p_t and the state's value estimate v_t. These outputs are instrumental in guiding the agent's action selection process throughout its MCTS. Lastly the agent selects or samples the best action a_t following the searched visit count distribution. During the *training* phase, given a training sequence $\{o_{t-l:t+K}, a_{t+1:t+K}, u_{t+1:t+K}, \pi_{t+1:t+K}, z_{t+1:t+K}\}$ at time t sampled from the replay buffer, where u_{t+k} denotes the actual reward obtained from the environment, π_{t+k} represents the target policy obtained through MCTS during the agent-environment interaction, and z_{t+k} is the value target computed using *n-step bootstrapping* (Hessel et al., 2018). The representation model initially converts the sequence of observations $o_{t-l:t}$ into the latent state s_t^0. Subsequently, the dynamic model executes K latent space rollouts based on the sequence of actions $a_{t+1:t+K}$. The latent state derived after the k-th rollout is denoted as s_t^k, with the corresponding predicted reward indicated as r_t^k. Upon receiving s_t^k, the prediction model generates a predicted policy p_t^k and a estimated value v_t^k. The final training loss encompasses three components: the policy loss (l_p), the value loss (l_v), and the reward loss (l_r):

$$L_{\text{MuZero}} = \sum_{k=0}^{K} l_p(\pi_{t+k}, p_t^k) + \sum_{k=0}^{K} l_v(z_{t+k}, v_t^k) + \sum_{k=1}^{K} l_r(u_{t+k}, r_t^k) \tag{2}$$

MuZero *Reanalyze*, as introduced in (Schrittwieser et al., 2021), is an advanced iteration of the original MuZero algorithm. This variant enhances the model's accuracy by conducting a fresh Monte Carlo Tree Search on sampled states with the most recent version of the model, subsequently utilizing the refined policy from this search to update the policy targets. Such reanalysis yields targets of superior quality compared to those obtained during the initial data collection phase. The Schrittwieser et al. (2021) expands upon this approach, formalizing it as a standalone method for policy refinement. This innovation opens avenues for its application in offline settings, where interactions with the environment are not possible. Traditional uses of the algorithm have intertwined reanalysis with training, periodically revisiting a subset of samples within a mini-batch for reevaluation. Furthermore, we suggests a paradigm shift, advocating for the decoupling of the reanalyze process from training routines, thus providing a more flexible and potentially more efficient methodology.

3.2. Stochastic Bandits

In the upcoming Section 4.2.2, we plan to offer an in-depth look at the theoretical assurances behind the performance of our newly designed ReZero agent, which is rooted in the fundamentals of bandit theory. Before diving into that, we'll start with an easy-to-understand overview of the key concepts surrounding Stochastic Bandits.

A stochastic bandit ν with K arms (Simchi-Levi et al., 2023) can be formulated as $\nu = \{P_i | i \in 1, 2, ..., K\}$, where P_i denotes the reward distribution corresponding to the arm i. The expected reward of arm i is denoted as $\mu_i(\nu)$. Given an algorithm π interacting with the bandit ν for n rounds, we define the regret R_n as follows:

$$R_n(\pi, \nu) = n\mu^*(\nu) - \mathbb{E}[\sum_{t=1}^{n} X_t] \tag{3}$$

where $\mu^*(\nu) = \max_{a\in[K]} \mu_a(\nu)$, X_t denotes the random reward achieved at time t.

The regret quantifies the loss of reward incurred by π compared to the optimal policy on ν during n rounds of interactions. In practical problems, we aim for π to exhibit sub-linear regret growth across a category of bandits Ω , i.e.:

$$\lim_{n\to\infty} \frac{R_n(\pi, \nu)}{n} = 0, \quad \text{for all } \nu \in \Omega \tag{4}$$

where sub-linear growth rate indicates π is choosing the optimal arm almost all of the time with n tends to infinity. UCB (Auer et al., 2002) is a classical bandit algorithm that achieves regret growth rate of $O(\log(n))$.

AlphaZero (Silver et al., 2017) considers the child nodes in tree search as bandit arms and applies a variant of the UCB in the tree search process:

$$UCB_{score}(s, a) = Q(s, a) + cP(s, a)\frac{\sqrt{\sum_b N(s, b)}}{1 + N(s, a)} \tag{5}$$

where s is the state corresponding to the current node, a is the action corresponding to a child node. $Q(s, a)$ is the mean return of choosing action a, $P(s, a)$ is the prior score of action a, $N(s, a)$ is the total time that a has been chosen, $\sum_b N(s, b)$ is the total time that s has been visited.

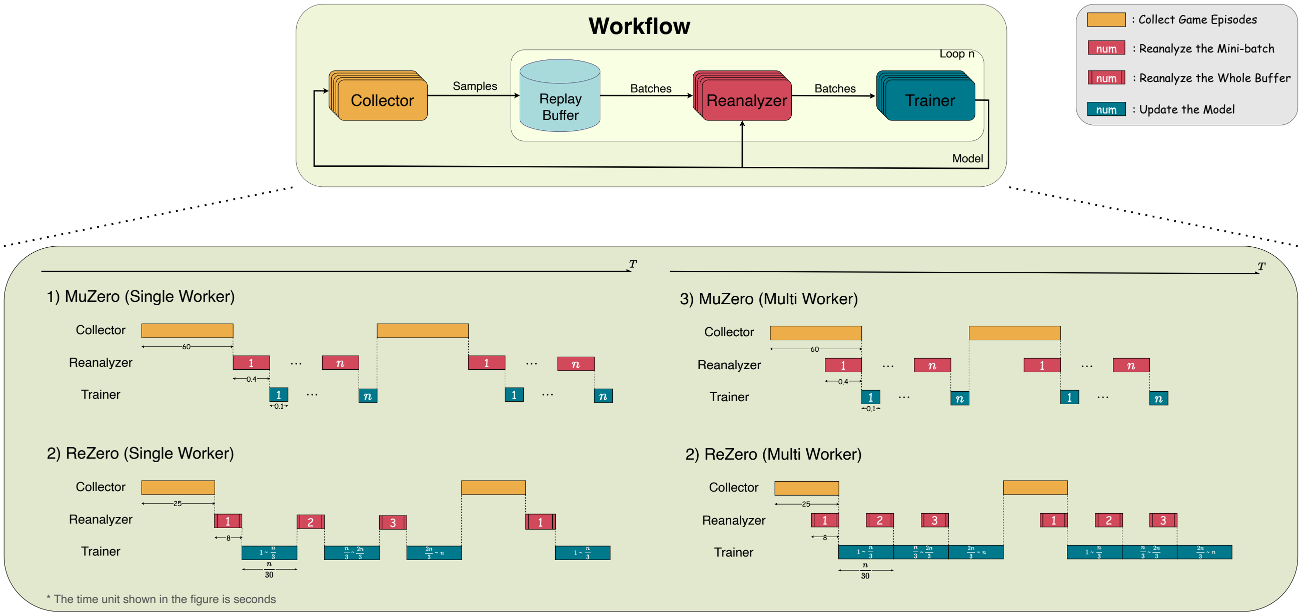

Figure 1. Execution workflow and runtime cycle graph about MuZero and ReZero in both single and multiple worker cases. The number inside the modules represent the number of iterations, and the number under the modules represent the time required for module execution. The model is updated n iterations between two collections. MuZero reanalyzes the mini-batch before each model update. ReZero reanalyzes the entire buffer after certain update iterations($\frac{n}{3}$ for example), which not only reduces the total number of MCTS calls, but also takes advantage of the processing speed of large batches.

4. Method

In this section, we introduce two techniques to boost MCTS-based algorithms. Section 4.1 describes the design of just-in-time reanalyze, which achieves sufficient training on policy target meanwhile significantly reduces the number of MCTS executions. Furthermore, section 4.2 introduces speedy reanalyze: an acceleration method based on information reuse, which utilizes search results from the subsequent time-step to speed up the search at the previous time-step, thereby saving the time required for a single MCTS execution.

4.1. Just-in-Time Reanalyze: A Simplified Framework

Reanalyze, as a method to enhance the quality of training targets, was previously bound together with the training process in earlier implementations (Schrittwieser et al., 2021) (Figure 1). However, recomputing the target policy at each training iteration incurs massive MCTS computations. This dilemma motivates us to find a method to reduce the amount of MCTS calculations while ensuring performance.

We begin our detailed analysis with the following perspective: consider the *reanalyze* process as a policy improvement operator $\mathcal{R}_{\theta^t}$, which takes the current policy P_{θ^t} and transforms it into the target policy T_{θ^t} for training.

$$T_{\theta^t}(a|s) = (\mathcal{R}_{\theta^t} P_{\theta^t})(a|s) = \frac{E_{T_a \sim \theta^t}[T_a]}{N} \tag{6}$$

where N is the number of simulations during the tree search and $E[T_a]$ is the expected total number that action a was chosen. During the *training* process, we update the model parameter θ according to the following equation:

$$\theta^{t+1} = \arg\min_{\theta} E_{s\sim\mathcal{D}}[\sum_{k=0}^{K} l^p(T_{\theta^t}(\cdot|s^k)), P_\theta(\cdot|s^k)) + l_{reg}] \tag{7}$$

where s is the state sampled from buffer $\mathcal{D}$, and s^k is the state after k unroll steps. We consider l_r and l_v in Equation 2 as a regularizer for l_p, and mark them as l_{reg}. In this way, the training can be seen as a projection operator $\mathcal{P}$ (Grill et al., 2020), projecting T_{θ^t} onto the policy space. The alternation of the *reanalyze* process and the *training* process constitutes policy iteration (Sutton & Barto, 1988), which can be seen as the composition of two operators:

$$P_{\theta^{t+1}} = (\mathcal{P} \circ \mathcal{R}_{\theta^t})P_{\theta^t} \tag{8}$$

In previous work (Schrittwieser et al., 2021), the *reanalyze* process is executed on the mini-batch of each training iteration. Changing the projection target frequently may result in an insufficient optimization, thereby degrading the quality of $\mathcal{P}$. In this way, even if the improvement operator $\mathcal{R}_\theta$ can produce high-quality training targets, the composite operator may not perform well. Therefore, we can adopt the practice of periodically updating the target network in DQN (Mnih et al., 2013), using *Just-in-Time Reanalyze* to change the projection targets only after several rounds of training.

Specifically, we propose the periodical reanalyze mechanism, i.e., we reanalyze the whole buffer after a fixed number of iterations. For each training iteration, we only need to sample the mini-batch and execute the gradient descent. Compared with previous methods, our method greatly reduces the number of MCTS, and can improve the parallelism due to the bigger search batch (Keskar et al., 2016), further enhanced the overall speed. Since the *reanalyze* process provides the target policy, the tree search during the collection phase is no longer necessary in the entire pipeline. Thus, we can bypass the tree search and directly select actions based on the policy, further reducing MCTS executions.

Based on our simplified framework, there are several potential advantages that may be explored in future work:

- When directly using policy for data collection, action selection is no longer constrained by the tree search. As a result, previous vectorized environments such as (Weng et al., 2022) can be integrated to speed up. Besides, this design also makes MCTS-based algorithms compatible to existing RL exploration methods like (Badia et al., 2020).
- Our method no longer needs to reanalyze the mini-batch for each iteration, thus decoupling the process of *reanalyze* and *training*. This provides greater scope for parallelization. In the case of multiple workers, we can design efficient parallelization paradigms as shown in Figure 1.

4.2. Speedy Reanalyze: Temporal Information Reuse

The above framework reduces the total number of MCTS. Next, we propose an temporal information reuse method to speed up the single tree search (Hereafter, termed *information reuse* for brevity). Section 4.2.1 introduces motivations and specific details. Section 4.2.2 carries out theoretical analysis of the method based on the bandit model.

4.2.1. Reuse the Information from Backward

As a local planning approach (Świechowski et al., 2023), MCTS is executed when receives each environment's state. The agent, by performing tree search on the current state, can gain the root value and the visit counts: the former tells the quality of the state, and the later tells which actions are more likely to yield good returns. These can be seen as a kind of **information**. The trajectories traversed by the search of adjacent time-steps often have many overlaps, indicating that it involves similar knowledge. **If the information gained at one time-step can be passed on to another, the search process can be accelerated**: for example, if the search conducted at time-step A reveals the quality of state s, then at time-step B, the agent need not to spend extra search cost to explore state s. Therefore, we designed a method to speed up the reanalyze process by reusing search information.

Algorithm 1 shows the specific design with Python-like code. During reanalyze, since the trajectories was already collected in the replay buffer, we can perform the search in a reverse order. That is to say, we first perform tree search

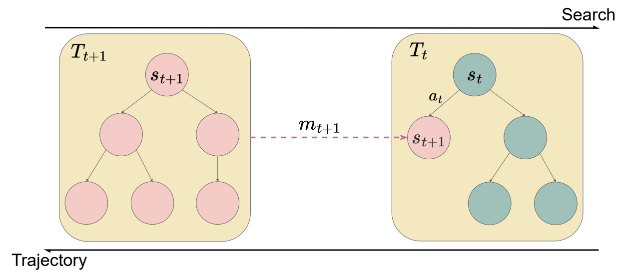

Figure 2. An illustration about the information reuse between adjacent time-steps. We search in the reverse order of the trajectory. After performing search on s_{t+1}, we obtain the search tree T_{t+1} and the root value m_{t+1}. When searching on s_t, we set the value of the node corresponding to s_{t+1} in search tree T_t as m_{t+1}. If action a_t is selected, the traverse process will be terminated, otherwise the traverse will proceed normally.

on the state of the last time-step in the trajectory, and then search the second to last, and so on. After we search on state s_{t+1} [1] (the state of the $t+1$-th time-step in the trajectory), the root value m_{t+1} is obtained. When performing search on s_t, we set the value of s_{t+1} to the fixed value m_{t+1}. During traverse in the tree, we select the action a_{root} for root node s_t according to the following equation:

$$a_{root} = \arg\max_a I_t(a) \tag{9}$$

$$I_t(a) = \begin{cases} UCB_{score}(s_t, a), & a \neq a_t \\ r_t + \gamma m_{t+1}, & a = a_t \end{cases} \tag{10}$$

where a is an action corresponding to a child node, a_t is the action corresponding to s_{t+1}, and r_t is the reward predicted by the dynamic model. If an action other than a_t is selected, the simulation just traverses following the original setting in MCTS (Schrittwieser et al., 2019). *If action a_t is selected, this simulation is terminated immediately.* Since the time used to search for node s_{t+1} is saved, the modified search process is faster than the original version, achieving *Speedy Reanalyze*.

4.2.2. Theoretical Analysis

AlphaZero employs a variant of the UCB algorithm during MCTS search. Subsequently, MuZero proposes further improvements to this variant formula. Practical evidence has demonstrated the effectiveness of both these node selection methods; however, they have not been theoretically analyzed in their original paper. In earlier studies, UCT analyzed the convergence properties of the UCB1 algorithm in a tree structure, albeit without introducing a prior policy. PUCB (Rosin, 2011) incorporates relevant analysis of a prior policy, but it uses the prior in a penalty function-like approach which is different from in the MCTS-based

[1]To facilitate the explanation, we have omitted details such as the latent space and do not make a deliberate distinction between states and nodes.

Algorithm 1 Python-like code for information reuse

```
# trajectory_segment: a segment with length K
def search_backwards(trajectory_segment):
   # prepare search context from the segment
   roots, actions = prepare(trajectory_segment)
   policy_targets = []
   # search the roots backwards
   for i in range(K, 1, -1):
      if i == K:
         # origin MCTS for Kth root
         policy, value = origin_MCTS(roots[i])
      else:
         # reuse information from previous search
         policy, value = reuse_MCTS(
            roots[i], actions[i], value
         )
      policy_targets.append(policy)

# N: simulation numbers during one search
def reuse_MCTS(root, action, values)
   for i in range(N):
      # select an action for root node
      a = select_root_child(root, action, value):
      # early stop the simulation
      if a == action:
         backpropagate()
         break
      # traverse to the leaf node
      else:
         traverse()
         backpropagate()
```

algorithm family. In recent work, Gumbel MuZero (Danihelka et al., 2022) approximated the tree search model as a stochastic bandit, and on this basis, proved the monotonic improvement of their designs. In this section, we will draw upon such a simplification and demonstrate within the context of a stochastic bandit model why the node selection formula and policy updating methods used in the MCTS-based algorithm family are effective, and how information reuse can influence the policy target.

Definition 4.1. After n simulations with a stochastic bandit ν with K arms, let T_i denote the number of visits to arm i, then the policy generated based on the visit distribution is a random vector $P = [\frac{T_1}{n}, ..., \frac{T_K}{n}]$. The value $V_\nu(P)$ of policy P on ν is defined as:

$$V_\nu(P) = \sum_i \frac{T_i \mu_i(\nu)}{n} \tag{11}$$

In cases where it does not cause any ambiguity, we will omit the mark ν in the following text. The next theorem explains the rationale behind using distribution as a target policy:

Theorem 4.2. *If an algorithm π exhibits sub-linear growth in regret R_n with respect to a bandit ν, then the expectation $E[V(P)]$ converges to μ^* as n tends to infinity.*

$$E[V(P)] = \sum_i \frac{E[T_i]\mu_i}{n} \tag{12}$$

$$= \sum_i \frac{E[T_i](\mu^* - \Delta_i)}{n} \tag{13}$$

$$= \mu^* - \frac{\sum_i E[T_i]\Delta_i}{n} \tag{14}$$

where $\Delta_i = \mu^* - \mu_i$, and $\sum_i E[T_i]\Delta_i$ is the regret R_n of the algorithm (see Appendix A).

Since R_n growths in a sub-linear rate, so we can get

$$\lim_{n\to\infty} E[V(P)] = \mu^* - \lim_{n\to\infty} \frac{R_n}{n} = \mu^* \tag{15}$$

The following theorem provides an upper bound for the growth rate of R_n for choosing arm in AlphaZero's way , thereby demonstrating that the regret is sub-linearly increasing.

Theorem 4.3. *Denote the prior for arm i as P_i. Without loss of generality, suppose $\mu^* = \mu_1$. Assume that the reward distribution of each arm i is 1-subgaussian and $P_1 > \mu_1$, then the regret R_n of AlphaZero satisfies (Appendix A)*

$$R_n \leq \sum_{i:\Delta_i>0} \inf_{\varepsilon\in(0,\Delta_i)} \Delta_i(3+\frac{16}{\varepsilon^4 P_1^2}+\frac{2}{(\Delta_i-\varepsilon)^2}+\frac{2P_i\sqrt{n-1}}{\Delta_i-\varepsilon}) \tag{16}$$

The upper bound given by Theorem 4.3 is $O(\sqrt{n})$, which, while not reaching the level of $O(\log n)$, is still sub-linear. Inequality (16) also indicates that a better policy which has higher probability to choose the best arm can lower the upper bound. Combining 4.2 and 4.3, we can conclude that with the increase in the number of simulations, the visit distribution will increasingly concentrate on the optimal arm, and the value corresponding to the policy will also converge to the value of the optimal arm. If the assumption that $P_1 > \mu_1$ doesn't hold, an upper bound with $O(\sqrt{n})$ rate can still be obtained by similar proof as long as there exists a t that $P_1\sqrt{t-1} > \mu_1$, which indicates unlike some algorithms that are sensitive to prior policies, MCTS has strong robustness to priors. As long as the number of simulations is sufficient, it can still achieve a good target policy.

When reusing information in the search process, we can obtain a reference value for an arm. The existence of this reference value can eliminate the need for exploration of known arms, thereby further reducing the algorithm's regret. The following theorem proves this point.

Theorem 4.4. *For a bandit which satisfies the assumptions in 4.3 and an already known arm value, the regret upper bound can be reduced if we use the way of 4.2.1 to choose arm. Specifically, if the known arm is the optimal arm with value μ_1, then the regret R_n satisfies:*

$$R_n \leq \sum_{i:\Delta_i>0} \inf_{\varepsilon\in(0,\Delta_i)} \Delta_i(2+\frac{2}{(\Delta_i-\varepsilon)^2}+\frac{2P_i\sqrt{n-1}}{\Delta_i-\varepsilon}) \tag{17}$$

In other case, with the known value μ_l of a sub-optimal arm

l, the regret R_n satisfies:

$$R_n \leq \sum_{i:\Delta_i>0, i\neq l} \inf_{\varepsilon\in(0,\Delta_i)} \Delta_i (3 + \frac{16}{\varepsilon^4 P_1^2} \tag{18}$$

$$+ \frac{2}{(\Delta_i - \varepsilon)^2} + \frac{2P_i\sqrt{n-1}}{\Delta_i - \varepsilon}) \tag{19}$$

$$+ \inf_{\varepsilon\in(0,\Delta_l)} \Delta_l (1 + \frac{16}{\varepsilon^4 P_1^2}) \tag{20}$$

The complete proof can be found in Appendix A.

From the proof, it can be seen that if the real value μ_1 of the optimal arm is known in advance, the regret caused by the underestimation of the optimal arm can be eliminated. However, the part of the regret that is reduced is $O(1)$. If the real value μ_i of a sub-optimal arm l is known in advance, the regret caused by continuous exploration of this sub-optimal arm can be eliminated. The part of the reduced regret is $O(\sqrt{n})$. Intuitively, it can be understood this way: If the value of a sub-optimal arm l is known, then after sufficient exploration of the optimal arm, the upper confidence bound will converge to be higher than μ_l with probability 1, thus making the regret caused by arm l no longer increases with n. While if the value of the optimal arm is known, then as the number of simulations n increases, there will always be a moment when the upper confidence bound of the sub-optimal arm i is greater than μ_1, so arm i will still be repeatedly selected, leading to the regret caused by arm i increases with n. However, in this case, since the optimal arm is chosen the most time as n increases, we can save more time by early stopping the search.

5. Experiment

Efficiency in RL usually refers to two aspects: *sample efficiency*, the agent's ability to learn effectively from a limited number of environmental interactions, gauged by the samples needed to reach a successful policy using equal compute resources; *time efficiency*, which is how swiftly an algorithm learns to make optimal decisions, indicated by the wall-clock time taken to achieve a successful policy. **Our primary goal is to improve time efficiency without compromising sample efficiency**. To empirically validate the efficiency of the two boosting techniques introduced in Section 3.2 for MCTS-based algorithms, we have incorporated the ReZero with two prominent algorithms detailed in (Niu et al., 2023): MuZero with a Self-Supervised Learning loss (SSL) and EfficientZero, yielding the enhanced variants ReZero-M and ReZero-E respectively. For brevity, when the context is unambiguous, we simply refer to them as ReZero, omitting the symbol representing the baseline algorithm.

In order to validate the applicability of our method across multiple types of decision-making environments, we opted for six representative *Atari* environments characterized by classic image-input observation and discrete action spaces, in addition to the strategic board games *Connect4* and *Gomoku* with special state spaces. For a baseline comparison, we employed the original implementations of MuZero and EfficientZero as delineated in the LightZero benchmark (Niu et al., 2023). The full implementation details available in Appendix B. Besides, we emphasize that to ensure an fair comparison of wall-clock time, all experimental trials were executed on a fixed single worker hardware settings.

In the next subsections, we sequentially explore and try to answer the following three questions:

- How much can ReZero-M/ReZero-E improve time efficiency compared to MuZero/EfficientZero, while maintaining equivalent levels of high sample efficiency?
- What is the concrete effect and hyper-parameter sensitivity of the *Just-in-Time Reanalyze* technique in ReZero?
- How much search budgets and time costs can be saved in practice by the *Temporal Information Reuse* technique?

5.1. Time Efficiency

Setup In this section, we aim to evaluate the performance of ReZero against classical MCTS-based algorithms MuZero and EfficientZero, focusing on the wall-clock time reduction required to achieve the comparable performance level. In order to facilitate a fair comparison in terms of wall-time, for each run, we not only utilize identical computational resources but also maintain consistent hyper-parameter settings across the algorithms (unless otherwise specified cases). Key parameters are aligned with those from original papers. Specifically, we set the *replay ratio* (the ratio between environment steps and training steps) (D'Oro et al., 2022; Schwarzer et al., 2023) to 0.25, and the *reanalyze ratio* (the ratio between targets computed from interactions with the environment and by reanalysing existing data) (Schrittwieser et al., 2021) is set to 1. For detailed hyper-paramater configurations, please refer to the Appendix B.

Results Figure 3 illustrates the performance comparison in terms of wall-clock time between ReZero-M and the original MuZero across a variety of RL environments. It is apparent that ReZero-M achieves a significant improvement in time efficiency, reaching a near-optimal policy in substantially less time for the games *Pong*, *Breakout* and *Seaquest*, and obtained a better policy than MuZero on the remaining tasks. Additional results about ReZero-E and EfficientZero are detailed in Appendix C.2, specifically in Figure 6 and 7. For a more comprehensive understanding of our findings, we provide an in-depth comparison of the wall-clock training time up to 100k environment steps for ReZero-M against MuZero in Table 1. The data reveals that, on most games, ReZero require between two to four times less wall-clock time per 100k steps compared to the baseline methods.

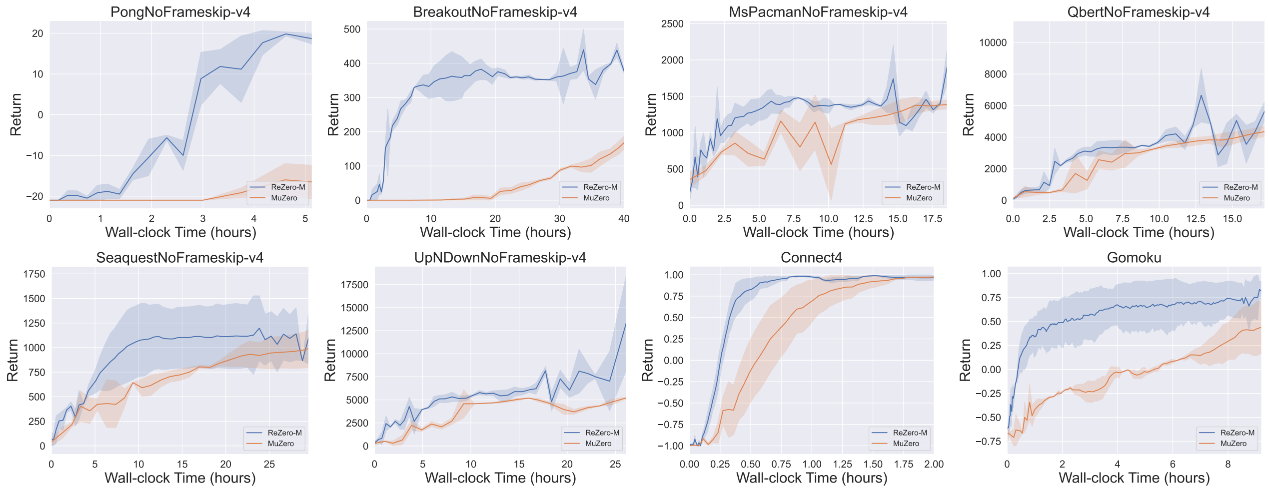

Figure 3. **Time-efficiency** of ReZero-M vs. MuZero on six representative *Atari* games and two board games (*Connect4*, *Gomoku*). The horizontal axis represents *Wall-clock Time* (hours), while the vertical axis indicates the *Episode Return* over 5 episodes. ReZero-M achieves higher time-efficiency than the baseline on most Atari games and *Gomoku*, and achieve similar time-efficiency on some simpler game like *Connect4*. Mean of 5 runs, shaded areas are 95% confidence intervals.

	Atari						Board Games	
avg. wall time (h) to 100k env. steps ↓	Pong	Breakout	MsPacman	Qbert	Seaquest	UpNDown	Connet4	Gomoku
ReZero-M (ours)	$\mathbf{1}_{\pm 0.6}$	$\mathbf{7.7}_{\pm 0.3}$	$\mathbf{3.8}_{\pm 0.2}$	$\mathbf{5.1}_{\pm 0.3}$	$\mathbf{6.1}_{\pm 0.5}$	$\mathbf{6.9}_{\pm 0.5}$	$\mathbf{10}_{\pm 0.8}$	$\mathbf{9}_{\pm 0.8}$
MuZero (Schrittwieser et al., 2019)	$4.1_{\pm 0.2}$	$12.4_{\pm 0.8}$	$12.0_{\pm 0.9}$	$10.2_{\pm 0.5}$	$11.75_{\pm 0.9}$	$12.4_{\pm 0.9}$	$11_{\pm 0.9}$	$23_{\pm 2}$

Table 1. **Average wall-time** of ReZero-M vs. MuZero on various tasks. *(left)* Six Atari games, *(right)* two board games. The time represents the average total wall-time to 100k environment steps for each algorithm. Mean and standard deviation over 5 runs.

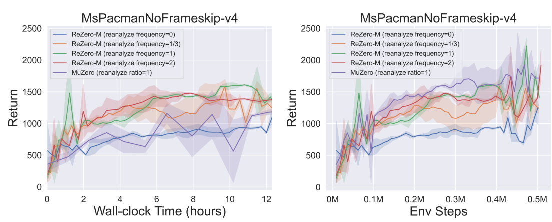

Figure 4. The effect of **Reanalyze Frequency** in ReZero-M on *MsPacman*. The horizontal axis in left represents *Wall-clock Time* (hours) while right side uses *Env Steps*. The vertical axis indicates *Return*. Proper *reanalyze frequency* can improve efficiency while obtaining the comparable *Return* with MuZero (*reanalyze ratio*=1).

Indicators	Avg. time (ms)	tree search (num calls)	dynamics model (num calls)	data process (num calls)
ReZero-M	$\mathbf{0.69 \pm 0.02}$	**6089**	**122**	**277**
MuZero	1.08 ± 0.09	13284	256	455

Table 2. Comparisons about detailed indicators between ReZero-M and MuZero inside the tree search. The number of function calls is the cumulative value of 100 training iterations on *Pong*. *Avg. time* is the average time of a MCTS across all calls.

5.2. Effect of Reanalyze Frequency

In next two sub-sectons, we will delve into details to explain how ReZero works. Firstly, we will analyze how the *Just-in-Time Reanalyze* technique manages to simplify the original iterative mini-batch reanalyze scheme to the periodical updated version while maintaining high sample efficiency. Here we adjust the periodical reanalyze frequency(reanalyze the buffer 1 time every n training iterations) in ReZero to a lower or higher value on the *MsPacman* environment, specifically *reanalyze frequency*=$\{2, 1, 1/3, 0\}$, where 1 denotes the setting $n = 1000$. The original MuZero variants with the *reanalyze ratio* of 1 is also included in this ablation experiment as a baseline. Figure 4 shows entire training curves in terms of *Wall-time* or *Env Steps* and validates that appropriate reanalyze frequency can save the time overhead without causing any obvious performance loss.

5.3. Effect of Information Reuse

To further validate and understand the advantages and significance of the *Information Reuse* technique proposed in ReZero, we meticulously document a suite of statistical indicators of the tree search process in Table 2. Comparative analysis between ReZero-M and MuZero reveals that the *Information Reuse* technique reduces the invocation frequency of the dynamics model, the search tree, and other operations like data process transformations. Consequently, this advanced technique in leveraging subsequent time step data contributes to save the tree search time in various MCTS-based algorithms, further leading to overall wall-clock time gains. Besides, *Information Reuse* also incrementally reallocates computations (tree search simulations) towards previously non-visited nodes rather than repeatedly accessing extant nodes, which maybe yield some performance gains in some environments (e.g. Breakout in Figure 3).

6. Conclusion and Limitation

In this paper, we have delved into the efficiency and scalability of MCTS-based algorithms. Unlike most existing works, we incorporates information reuse and periodic reanalyze techniques to reduces wall-clock time costs while preserving sample efficiency. Our theoretical analysis and experimental results confirm that ReZero efficiently reduces the time cost and maintains or even improves performance across different decision-making domains. However, our current experiments are mainly conducted on the single worker setting, there exists considerable optimization scope to apply our approach into distributed RL training, and our design harbors the potential of better parallel acceleration and more stable convergence in large-scale training tasks. Also, the combination between ReZero and frameworks akin to AlphaZero, or its integration with some recent offline datasets such as RT-X (Padalkar et al., 2023), constitutes a fertile avenue for future research. These explorations could broadening the application horizons of MCTS-based algorithms.

7. Impact Statements

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.

8. Acknowledgements

Thanks to each of the authors for their efforts in the work of ReZero. We spent several memorable midnights and early mornings together in the lab. Thanks to everyone at OpenDILab and their pioneering work, especially the LightZero benchmark (`https://github.com/opendilab/LightZero`) for providing the base code for ReZero. Finally, we would like to thank Shanghai AI Laboratory and professor Jing Yang of Xi'an Jiaotong University for providing a platform for this work.

References

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.

Auer, P., Cesa-Bianchi, N., and Fischer, P. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47:235–256, 2002.

Badia, A. P., Sprechmann, P., Vitvitskyi, A., Guo, D., Piot, B., Kapturowski, S., Tieleman, O., Arjovsky, M., Pritzel, A., Bolt, A., et al. Never give up: Learning directed exploration strategies. *arXiv preprint arXiv:2002.06038*, 2020.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

Burda, Y., Edwards, H., Storkey, A., and Klimov, O. Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*, 2018.

Danihelka, I., Guez, A., Schrittwieser, J., and Silver, D. Policy improvement by planning with gumbel. In *International Conference on Learning Representations*, 2022.

D'Oro, P., Schwarzer, M., Nikishin, E., Bacon, P.-L., Bellemare, M. G., and Courville, A. Sample-efficient reinforcement learning by breaking the replay ratio barrier. In *Deep Reinforcement Learning Workshop NeurIPS 2022*, 2022.

Fu, Y., Sun, M., Nie, B., and Gao, Y. Accelerating monte carlo tree search with probability tree state abstraction. *arXiv preprint arXiv:2310.06513*, 2023.

Grill, J.-B., Altché, F., Tang, Y., Hubert, T., Valko, M., Antonoglou, I., and Munos, R. Monte-carlo tree search as regularized policy optimization. In *International Conference on Machine Learning*, pp. 3769–3778. PMLR, 2020.

Grimm, C., Barreto, A., Singh, S., and Silver, D. The value equivalence principle for model-based reinforcement learning. *Advances in Neural Information Processing Systems*, 33:5541–5552, 2020a.

Grimm, C., Barreto, A., Singh, S., and Silver, D. The value equivalence principle for model-based reinforcement learning. *Advances in Neural Information Processing Systems*, 33:5541–5552, 2020b.

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

Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Hubert, T., Schrittwieser, J., Antonoglou, I., Barekatain, M., Schmitt, S., and Silver, D. Learning and planning in complex action spaces. In Meila, M. and Zhang, T. (eds.), *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July*

2021, Virtual Event, volume 139 of *Proceedings of Machine Learning Research*, pp. 4476–4486. PMLR, 2021. URL http://proceedings.mlr.press/v139/hubert21a.html.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pp. 448–456. pmlr, 2015.

Janner, M., Fu, J., Zhang, M., and Levine, S. When to trust your model: Model-based policy optimization. *Advances in neural information processing systems*, 32, 2019.

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P. T. P. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *ICLR*, 2015.

Langley, P. Crafting papers on machine learning. In Langley, P. (ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

Leurent, E. and Maillard, O.-A. Monte-carlo graph search: the value of merging similar states. In *Asian Conference on Machine Learning*, pp. 577–592. PMLR, 2020.

Li, Q., Peng, Z., Feng, L., Zhang, Q., Xue, Z., and Zhou, B. Metadrive: Composing diverse driving scenarios for generalizable reinforcement learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

Mei, Y., Gao, J., Ye, W., Liu, S., Gao, Y., and Wu, Y. Speedyzero: Mastering atari with limited data and time. In *The Eleventh International Conference on Learning Representations*, 2023.

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

Niu, Y., Pu, Y., Yang, Z., Li, X., Zhou, T., Ren, J., Hu, S., Li, H., and Liu, Y. Lightzero: A unified benchmark for monte carlo tree search in general sequential decision scenarios. *arXiv preprint arXiv:2310.08348*, 2023.

Padalkar, A., Pooley, A., Jain, A., Bewley, A., Herzog, A., Irpan, A., Khazatsky, A., Rai, A., Singh, A., Brohan, A., et al. Open x-embodiment: Robotic learning datasets and rt-x models. *arXiv preprint arXiv:2310.08864*, 2023.

Pohlen, T., Piot, B., Hester, T., Azar, M. G., Horgan, D., Budden, D., Barth-Maron, G., Van Hasselt, H., Quan, J., Večerík, M., et al. Observe and look further: Achieving consistent performance on atari. *arXiv preprint arXiv:1805.11593*, 2018.

Rafailov, R., Sharma, A., Mitchell, E., Ermon, S., Manning, C. D., and Finn, C. Direct preference optimization: Your language model is secretly a reward model. *arXiv preprint arXiv:2305.18290*, 2023.

Rosin, C. D. Multi-armed bandits with episode context. *Annals of Mathematics and Artificial Intelligence*, 61(3): 203–230, 2011.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., Lillicrap, T. P., and Silver, D. Mastering atari, go, chess and shogi by planning with a learned model. *CoRR*, abs/1911.08265, 2019. URL http://arxiv.org/abs/1911.08265.

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., 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.

Simchi-Levi, D., Zheng, Z., and Zhu, F. Stochastic multi-armed bandits: Optimal trade-off among optimality, consistency, and tail risk. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

Sutton, R. S. and Barto, A. G. Reinforcement learning: An introduction. *IEEE Transactions on Neural Networks*, 16: 285–286, 1988.

Świechowski, M., Godlewski, K., Sawicki, B., and Mańdziuk, J. Monte carlo tree search: A review of recent modifications and applications. *Artificial Intelligence Review*, 56(3):2497–2562, 2023.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., Oh, J., Horgan, D., Kroiss, M., Danihelka, I., Huang, A., Sifre, L., Cai, T., Agapiou, J. P., Jaderberg, M., Vezhnevets, A. S., Leblond, R., Pohlen, T., Dalibard, V., Budden, D., Sulsky, Y., Molloy, J., Paine, T. L., Gülçehre, Ç., Wang, Z., Pfaff, T., Wu, Y., Ring, R., Yogatama, D., Wünsch, D., McKinney, K., Smith, O., Schaul, T., Lillicrap, T. P., Kavukcuoglu, K., Hassabis, D., Apps, C., and Silver, D. Grandmaster level in starcraft II using multi-agent reinforcement learning. *Nat.*, 575(7782):350–354, 2019. doi: 10.1038/s41586-019-1724-z. URL https://doi.org/10.1038/s41586-019-1724-z.

Weng, J., Lin, M., Huang, S., Liu, B., Makoviichuk, D., Makoviychuk, V., Liu, Z., Song, Y., Luo, T., Jiang, Y., et al. Envpool: A highly parallel reinforcement learning environment execution engine. *Advances in Neural Information Processing Systems*, 35:22409–22421, 2022.

Wu, D. J. Accelerating self-play learning in go. *arXiv preprint arXiv:1902.10565*, 2019.

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.

References

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.

Auer, P., Cesa-Bianchi, N., and Fischer, P. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47:235–256, 2002.

Badia, A. P., Sprechmann, P., Vitvitskyi, A., Guo, D., Piot, B., Kapturowski, S., Tieleman, O., Arjovsky, M., Pritzel, A., Bolt, A., et al. Never give up: Learning directed exploration strategies. *arXiv preprint arXiv:2002.06038*, 2020.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

Burda, Y., Edwards, H., Storkey, A., and Klimov, O. Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*, 2018.

Danihelka, I., Guez, A., Schrittwieser, J., and Silver, D. Policy improvement by planning with gumbel. In *International Conference on Learning Representations*, 2022.

D'Oro, P., Schwarzer, M., Nikishin, E., Bacon, P.-L., Bellemare, M. G., and Courville, A. Sample-efficient reinforcement learning by breaking the replay ratio barrier. In *Deep Reinforcement Learning Workshop NeurIPS 2022*, 2022.

Fu, Y., Sun, M., Nie, B., and Gao, Y. Accelerating monte carlo tree search with probability tree state abstraction. *arXiv preprint arXiv:2310.06513*, 2023.

Grill, J.-B., Altché, F., Tang, Y., Hubert, T., Valko, M., Antonoglou, I., and Munos, R. Monte-carlo tree search as regularized policy optimization. In *International Conference on Machine Learning*, pp. 3769–3778. PMLR, 2020.

Grimm, C., Barreto, A., Singh, S., and Silver, D. The value equivalence principle for model-based reinforcement learning. *Advances in Neural Information Processing Systems*, 33:5541–5552, 2020a.

Grimm, C., Barreto, A., Singh, S., and Silver, D. The value equivalence principle for model-based reinforcement learning. *Advances in Neural Information Processing Systems*, 33:5541–5552, 2020b.

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

Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Hubert, T., Schrittwieser, J., Antonoglou, I., Barekatain, M., Schmitt, S., and Silver, D. Learning and planning in complex action spaces. In Meila, M. and Zhang, T. (eds.), *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pp. 4476–4486. PMLR, 2021. URL http://proceedings.mlr.press/v139/hubert21a.html.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pp. 448–456. pmlr, 2015.

Janner, M., Fu, J., Zhang, M., and Levine, S. When to trust your model: Model-based policy optimization. *Advances in neural information processing systems*, 32, 2019.

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P. T. P. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *ICLR*, 2015.

Langley, P. Crafting papers on machine learning. In Langley, P. (ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

Leurent, E. and Maillard, O.-A. Monte-carlo graph search: the value of merging similar states. In *Asian Conference on Machine Learning*, pp. 577–592. PMLR, 2020.

Li, Q., Peng, Z., Feng, L., Zhang, Q., Xue, Z., and Zhou, B. Metadrive: Composing diverse driving scenarios for generalizable reinforcement learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

Mei, Y., Gao, J., Ye, W., Liu, S., Gao, Y., and Wu, Y. Speedyzero: Mastering atari with limited data and time. In *The Eleventh International Conference on Learning Representations*, 2023.

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

Niu, Y., Pu, Y., Yang, Z., Li, X., Zhou, T., Ren, J., Hu, S., Li, H., and Liu, Y. Lightzero: A unified benchmark for monte carlo tree search in general sequential decision scenarios. *arXiv preprint arXiv:2310.08348*, 2023.

Padalkar, A., Pooley, A., Jain, A., Bewley, A., Herzog, A., Irpan, A., Khazatsky, A., Rai, A., Singh, A., Brohan, A., et al. Open x-embodiment: Robotic learning datasets and rt-x models. *arXiv preprint arXiv:2310.08864*, 2023.

Pohlen, T., Piot, B., Hester, T., Azar, M. G., Horgan, D., Budden, D., Barth-Maron, G., Van Hasselt, H., Quan, J., Večerík, M., et al. Observe and look further: Achieving consistent performance on atari. *arXiv preprint arXiv:1805.11593*, 2018.

Rafailov, R., Sharma, A., Mitchell, E., Ermon, S., Manning, C. D., and Finn, C. Direct preference optimization: Your language model is secretly a reward model. *arXiv preprint arXiv:2305.18290*, 2023.

Rosin, C. D. Multi-armed bandits with episode context. *Annals of Mathematics and Artificial Intelligence*, 61(3): 203–230, 2011.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., Lillicrap, T. P., and Silver, D. Mastering atari, go, chess and shogi by planning with a learned model. *CoRR*, abs/1911.08265, 2019. URL http://arxiv.org/abs/1911.08265.

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., 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.

Simchi-Levi, D., Zheng, Z., and Zhu, F. Stochastic multi-armed bandits: Optimal trade-off among optimality, consistency, and tail risk. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

Sutton, R. S. and Barto, A. G. Reinforcement learning: An introduction. *IEEE Transactions on Neural Networks*, 16: 285–286, 1988.

Świechowski, M., Godlewski, K., Sawicki, B., and Mańdziuk, J. Monte carlo tree search: A review of recent modifications and applications. *Artificial Intelligence Review*, 56(3):2497–2562, 2023.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., Oh, J., Horgan, D., Kroiss, M., Danihelka, I., Huang, A., Sifre, L., Cai, T., Agapiou, J. P., Jaderberg, M., Vezhnevets, A. S., Leblond, R., Pohlen, T., Dalibard, V., Budden, D., Sulsky, Y., Molloy, J., Paine, T. L., Gülçehre, Ç., Wang, Z., Pfaff, T., Wu, Y., Ring, R., Yogatama, D., Wünsch, D., McKinney, K., Smith, O., Schaul, T., Lillicrap, T. P., Kavukcuoglu, K., Hassabis, D., Apps, C., and

Silver, D. Grandmaster level in starcraft II using multi-agent reinforcement learning. *Nat.*, 575(7782):350–354, 2019. doi: 10.1038/s41586-019-1724-z. URL https://doi.org/10.1038/s41586-019-1724-z.

Weng, J., Lin, M., Huang, S., Liu, B., Makoviichuk, D., Makoviychuk, V., Liu, Z., Song, Y., Luo, T., Jiang, Y., et al. Envpool: A highly parallel reinforcement learning environment execution engine. *Advances in Neural Information Processing Systems*, 35:22409–22421, 2022.

Wu, D. J. Accelerating self-play learning in go. *arXiv preprint arXiv:1902.10565*, 2019.

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.

A. Proof Materials

To verify the theoretical convergence guarantee of ReZero, we have provided some parts of detailed theoretical proofs in the main paper. In this section, we will present the complete supplementary proofs.

Lemma A.1. *Regret decomposition lemma:Denote X_t as the reward at time t, A_t as the arm chosen at time t, and $S_n = \sum_t X_t$, then the regret can be decomposed by regret caused by every sub-optimal arm:*

$$R_n = \sum_i E[T_i]\Delta_i \tag{21}$$

Proof. Note that

$$\begin{aligned} R_n &= n\mu^* - E[S_n] &(22)\\ &= \sum_i \sum_{t=1}^{n} E[(\mu^* - X_t)\mathbb{I}\{A_t = i\}] &(23)\end{aligned}$$

and

$$\begin{aligned} &E[(\mu^* - X_t)\mathbb{I}\{A_t = i\}|A_t] &(24)\\ &= \mathbb{I}\{A_t = i\}E[\mu^* - X_t|A_t] &(25)\\ &= \mathbb{I}\{A_t = i\}(\mu^* - \mu_{A_t}) &(26)\\ &= \mathbb{I}\{A_t = i\}(\mu^* - \mu_i) &(27)\\ &= \mathbb{I}\{A_t = i\}\Delta_i &(28)\\ & &(29)\end{aligned}$$

The proof can be completed by combining the two equations and using the definition of T_i. □

Lemma A.2. *Let $X_1, ..., X_n$ be a sequence of independent 1-subgaussian random variables, $\hat{\mu}_t = \frac{1}{t}\sum_{s=1}^{t} X_s$, $\varepsilon > 0$, $a > 0$ and*

$$\kappa = \sum_{t=1}^{n} \mathbb{I}\{\hat{\mu}_t + \sqrt{\frac{a}{t^2}} \geq \varepsilon\} \tag{30}$$

, then it holds that $E[\kappa] \leq 1 + \frac{2\sqrt{a}}{\varepsilon} + \frac{2}{\varepsilon^2}$.

Proof. Take u as $\frac{2\sqrt{a}}{\varepsilon}$, then

$$\begin{aligned} E[\kappa] &\leq u + \sum_{t=\lceil u \rceil}^{n} \mathbb{P}(\hat{\mu}_t \sqrt{\frac{a}{t^2}} \geq \varepsilon) &(31)\\ &\leq u + \sum_{t=\lceil u \rceil}^{n} \exp(-\frac{t(\varepsilon - \sqrt{\frac{a}{t^2}})^2}{2}) &(32)\\ &\leq 1 + u + \int_u^\infty \exp(-\frac{t(\varepsilon - \sqrt{\frac{a}{t^2}})^2}{2})dt &(33)\\ &\leq 1 + u + e^{\sqrt{a}\varepsilon}\int_u^\infty e^{-\frac{t\varepsilon^2}{2}}dt &(34)\\ &= 1 + u + \frac{2}{\varepsilon^2} = 1 + \frac{2\sqrt{a}}{\varepsilon} + \frac{2}{\varepsilon^2} &(35)\end{aligned}$$

□

Proof for Theorem 4.3:

Proof. Denote $T_i(k)$ as the times that arm i has been chosen until time k, $\hat{\mu}_{is}$ as the empirical mean based on the first s samples from distribution P_i, and $\hat{\mu}_i(k) = \hat{\mu}_{iT_i(k)}$. Then we have:

$$T_i(n) = \sum_{t=1}^{n} \mathbb{I}\{A_t = i\} \tag{36}$$

$$\leq \sum_{t=1}^{n} \mathbb{I}\{\hat{\mu}_1(t-1) + P_1 \frac{\sqrt{t-1}}{1+T_1(t-1)} \leq \mu_1 - \varepsilon\} \tag{37}$$

$$+ \sum_{t=1}^{n} \mathbb{I}\{\hat{\mu}_i(t-1) + P_i \frac{\sqrt{t-1}}{1+T_i(t-1)} \geq \mu_1 - \varepsilon \text{ and } A_t = i\} \tag{38}$$

for 37, we have

$$E[\sum_{t=1}^{n} \mathbb{I}\{\hat{\mu}_1(t-1) + P_1 \frac{\sqrt{t-1}}{1+T_1(t-1)} \leq \mu_1 - \varepsilon\}] \tag{39}$$

$$\leq 1 + \sum_{t=2}^{n} \sum_{s=0}^{t-1} \mathbb{P}(\hat{\mu}_{1s} + P_1 \frac{\sqrt{t-1}}{1+s} \leq \mu_1 - \varepsilon) \tag{40}$$

$$= 1 + \sum_{t=2}^{n} \sum_{s=1}^{t-1} \mathbb{P}(\hat{\mu}_{1s} + P_1 \frac{\sqrt{t-1}}{1+s} \leq \mu_1 - \varepsilon) \tag{41}$$

$$\leq 1 + \sum_{t=2}^{n} \sum_{s=1}^{t-1} \exp(-\frac{s(\varepsilon + P_1 \frac{\sqrt{t-1}}{1+s})^2}{2}) \tag{42}$$

$$\leq 1 + \sum_{t=2}^{n} \exp(-\frac{1}{2}\varepsilon P_1 \sqrt{t-1}) \sum_{s=1}^{t-1} \exp(-\frac{s\varepsilon^2}{2}) \tag{43}$$

$$\leq 1 + \sum_{t=2}^{n} \exp(-\frac{1}{2}\varepsilon P_1 \sqrt{t-1}) \frac{2}{\varepsilon^2} \tag{44}$$

$$\leq 1 + \frac{16}{\varepsilon^4 P_1^2} \tag{45}$$

Notes: In 40, since we assume $P_1 \geq \mu_1$, the probability of the term $s = 0$ would be 0. Thus, we can discard it. If this assumption doesn't hold, we can choose to accumulate t starting from a larger t_0(which satisfies $P_1\sqrt{t_0 - 1} > \mu_1$ as mentioned in the article). Starting the summation from such a t_0 ensures all terms of $s = 0$ can still be discarded, and all add terms that $t \leq t_0$ can be bounded to 1. This only changes the constant term and won't affect the growth rate of regret. From 42 to 43, we just need to expand the quadratic term and do some simple inequality scaling. From 43 to 44, we need to notice that $\sum_{s=1}^{t-1} \exp(-\frac{s\varepsilon^2}{2})$ is a geometric sequence and scale it to $\frac{2}{\varepsilon^2}$. From 44 to 45, we use the inequality $\sum_{t=2}^{n} \frac{1}{e^{a\sqrt{t-1}}} \leq \int_1^\infty \frac{1}{e^{a\sqrt{t-1}}}$.

for 38, we have

$$E[\sum_{t=1}^{n} \mathbb{I}\{\hat{\mu}_i(t-1) + P_i \frac{\sqrt{t-1}}{1+T_i(t-1)} \geq \mu_1 - \varepsilon \text{ and } A_t = i\}] \tag{46}$$

$$\leq E[\sum_{t=1}^{n} \mathbb{I}\{\hat{\mu}_i(t-1) + P_i \sqrt{\frac{n-1}{(1+T_i(t-1))^2}} \geq \mu_1 - \varepsilon \text{ and } A_t = i\}] \tag{47}$$

$$\leq 1 + E[\sum_{s=1}^{n-1} \mathbb{I}\{\hat{\mu}_{is} + P_i \sqrt{\frac{n-1}{(1+s)^2}} \geq \mu_1 - \varepsilon\}] \tag{48}$$

$$\leq 1 + E[\sum_{s=1}^{n} \mathbb{I}\{\hat{\mu}_{is} - \mu_i + P_i \sqrt{\frac{n-1}{s^2}} \geq \Delta_i - \varepsilon\}] \tag{49}$$

with Lemma A.2, we can have

$$49 \leq 2 + \frac{2P_i\sqrt{n-1}}{\Delta_i - \varepsilon} + \frac{2}{(\Delta_i - \varepsilon)^2} \tag{50}$$

so

$$E[T_i(n)] \leq 3 + \frac{2P_i\sqrt{n-1}}{\Delta_i - \varepsilon} + \frac{2}{(\Delta_i - \varepsilon)^2} + \frac{16}{\varepsilon^4 P_1^2} \tag{51}$$

combined with Lemma A.1, the proof is completed. □

Proof for Theorem 4.4:

Proof. The proof can be obtained by making slight modifications to the proof of 4.3. In case that μ_1 is already known,

$$T_i(n) = \sum_{t=1}^{n} \mathbb{I}\{A_t = i\} \tag{52}$$

$$\leq \sum_{t=1}^{n} \mathbb{I}\{\mu_1 \leq \mu_1 - \varepsilon\} \tag{53}$$

$$+ \sum_{t=1}^{n} \mathbb{I}\{\hat{\mu}_i(t-1) + P_i \frac{\sqrt{t-1}}{1 + T_i(t-1)} \geq \mu_1 - \varepsilon \text{ and } A_t = i\} \tag{54}$$

$$= \sum_{t=1}^{n} \mathbb{I}\{\hat{\mu}_i(t-1) + P_i \frac{\sqrt{t-1}}{1 + T_i(t-1)} \geq \mu_1 - \varepsilon \text{ and } A_t = i\} \tag{55}$$

$$\leq 2 + \frac{2P_i\sqrt{n-1}}{\Delta_i - \varepsilon} + \frac{2}{(\Delta_i - \varepsilon)^2} \tag{56}$$

In case that a sub-optimal value μ_l is known,

$$T_l(n) = \sum_{t=1}^{n} \mathbb{I}\{A_t = l\} \tag{57}$$

$$\leq \sum_{t=1}^{n} \mathbb{I}\{\hat{\mu}_1(t-1) + P_1 \frac{\sqrt{t-1}}{1 + T_1(t-1)} \leq \mu_1 - \varepsilon\} \tag{58}$$

$$+ \sum_{t=1}^{n} \mathbb{I}\{\mu_l \geq \mu_1 - \varepsilon \text{ and } A_t = i\} \tag{59}$$

$$= \sum_{t=1}^{n} \mathbb{I}\{\hat{\mu}_1(t-1) + P_1 \frac{\sqrt{t-1}}{1 + T_1(t-1)} \leq \mu_1 - \varepsilon\} \tag{60}$$

$$\leq 1 + \frac{16}{\varepsilon^4 P_1^2} \tag{61}$$

and the bound of $T_i(n)$ for $i \neq l$ keeps unchanged.

In both cases, combined with Lemma A.1, the proof is completed. □

B. Implementation Details

B.1. Environments

In this section, we first introduce various types of reinforcement learning environments evaluated in the main paper and their respective characteristics, including different observation/action/reward space and transition functions.

Atari: This category includes sub-environments like *Pong, Qbert, Ms.Pacman, Breakout, UpNDown*, and *Seaquest*. In these environments, agents control game characters and perform tasks based on pixel input, such as hitting bricks in *Breakout*. With their high-dimensional visual input and discrete action space features, Atari environments are widely used to evaluate the capability of reinforcement learning algorithms in handling visual inputs and discrete control problems.

Board Games: This types of environment includes *Connect4*, *Gomoku*, where uniquely marked boards and explicitly defined rules for placement, movement, positioning, and attacking are employed to achieve the game's ultimate objective. These environments feature a variable discrete action space, allowing only one player's piece per board position. In practice, algorithms utilize action mask to indicate reasonable actions.

B.2. Algorithm Implementation Details

Our algorithm's implementation is based on the open-source code of LightZero (Niu et al., 2023). Given that our proposed theoretical improvements are applicable to any MCTS-based RL method, we have chosen MuZero and EfficientZero as case studies to investigate the practical improvements in time efficiency achieved by integrating the ReZero boosting techniques: *just-in-time reanalyze* and *speedy reanalyze (temporal information reuse)*.

To ensure an equitable comparison of wall-clock time, all experimental trials were executed on a fixed single worker hardware setting consisting of a single NVIDIA A100 GPU with 30 CPU cores and 120 GiB memory. Besides, we emphasize that to ensure a fair comparison of time efficiency and sample efficiency, the model architecture and hyper-parameters used in the experiments of Section 5 are essentially consistent with the settings in LightZero (Niu et al., 2023). For specific hyper-parameters of *ReZero-M* and *MuZero* on Atari, please refer to the Table 3. The main different hyper-parameters for the *ReZero-M* algorithm in the *Connect4* and *Gomoku* environment are set out in Tables 4. In addition to employing an LSTM network with a hidden state dimension of 512 to predict the value prefix (Ye et al., 2021), all hyperparameters of ReZero-E are essentially identical to those of ReZero-M in Table 3.

Wall-time statistics Note that all our current tests are conducted in the single-worker case. Therefore, the wall-time reported in Table 1 and Appendix C.2 for reaching 100k env steps includes:

- *collect time*: The total time spent by an agent interacting with the environment to gather experience data.(Weng et al., 2022) can be integrated to speed up. Besides, this design also makes MCTS-based algorithms compatible to existing RL exploration methods like (Burda et al., 2018).
- *reanalyze time*: The time used to reanalyze collected data with the current policy or value function for more accurate learning targets (Schrittwieser et al., 2021).
- *train time*: The duration for performing updates to the agent's policy, value functions and model based on collected data.
- *evaluation time*: The period during which the agent's policy is tested against the environment, separate from training, to assess performance.

Currently, we have set *collect_max_episode_steps* to 10,000 and *eval_max_episode_steps* to 20,000 to mitigate the impact of anomalously long evaluation episodes on time. In the future, we will consider conducting offline evaluations to avoid the influence of evaluation time on our measurement of time efficiency. Furthermore, the ReZero methodology represents a pure algorithmic enhancement, eliminating the need for supplementary computational resources or additional overhead. This approach is versatile, enabling seamless integration with single-worker serial execution environments as well as multi-worker asynchronous frameworks. The exploration of ReZero's extensions and its evaluations in a multi-worker (Mei et al., 2023) paradigm are earmarked for future investigation.

Board games settings Given that our primary objective is to test the proposed techniques for improvements in time efficiency, we consider a simplified version of single-player mode in all the board games. This involves setting up a fixed but powerful expert bot and treating this opponent as an integral part of the environment. Exploration of our proposed techniques in the context of learning through self-play training pipeline is reserved for our future work.

Hyperparameter	Value
Replay Ratio (Schwarzer et al., 2023)	0.25
Reanalyze ratio (Schrittwieser et al., 2021)	1
Num of frames stacked	4
Num of frames skip	4
Reward clipping (Mnih et al., 2013)	True
Optimizer type	Adam (Kingma & Ba, 2015)
Learning rate	3×10^{-3}
Discount factor	0.997
Weight of policy loss	1
Weight of value loss	0.25
Weight of reward loss	1
Weight of policy entropy loss	0
Weight of SSL (self-supervised learning) loss (Ye et al., 2021)	2
Batch size	256
Model update ratio	0.25
Frequency of target network update	100
Weight decay	10^{-4}
Max gradient norm	10
Length of game segment	400
Replay buffer size (in transitions)	1e6
TD steps	5
Number of unroll steps	5
Use augmentation	True
Discrete action encoding type	One Hot
Normalization type	Batch Normalization (Ioffe & Szegedy, 2015)
Priority exponent coefficient (Schaul et al., 2015)	0.6
Priority correction coefficient	0.4
Dirichlet noise alpha	0.3
Dirichlet noise weight	0.25
Number of simulations in MCTS (sim)	50
Categorical distribution in value and reward modeling	True
The scale of supports used in categorical distribution (Pohlen et al., 2018)	300

Table 3. Key hyperparameters of **ReZero-M** on *Atari* environments.

C. Additional Experiments

C.1. Sample-efficiency of ReZero-M

Except for the results reported in main paper for time efficiency, we also shows the detailed sample efficiency results in this section. Figure 5 displays the performance over environment interaction steps of the ReZero-M algorithm compared with the original MuZero algorithm across six representative Atari environments and two board games. We can find that ReZero-M obtained *similar* sample efficiency than MuZero on the most tasks.

C.2. ReZero-E

The enhancements of ReZero we have proposed are universally applicable to any MCTS-based reinforcement learning approach theoretically. In this section, we integrate ReZero with EfficientZero to obtain the enhanced ReZero-E algorithm. We present the empirical results comparing ReZero-E with the standard EfficientZero across four Atari environments.

Figure 6 shows that ReZero-E is better than EfficientZero in terms of time efficiency. Figure 7 indicates that ReZero-E matches EfficientZero's sample efficiency across most tasks. Additionally, Table C.2 details training times to 100k environment steps, revealing that ReZero-E is *0.5 to 2 times faster* than baseline methods on most games.

Hyperparameter	**Value**
Replay Ratio (Schwarzer et al., 2023)	0.25
Reanalyze ratio (Schrittwieser et al., 2021)	1
Board size	6x7; 6x6
Num of frames stacked	1
Discount factor	1
Weight of SSL (self-supervised learning) loss	0
Length of game segment	18
TD steps	21; 18
Use augmentation	False
Number of simulations in MCTS (sim)	50
The scale of supports used in categorical distribution	10

Table 4. Key hyperparameters of **ReZero-M** on *Connect4* and *Gomoku* environments. If the parameter settings of these two environments are different, they are separated by a semicolon.

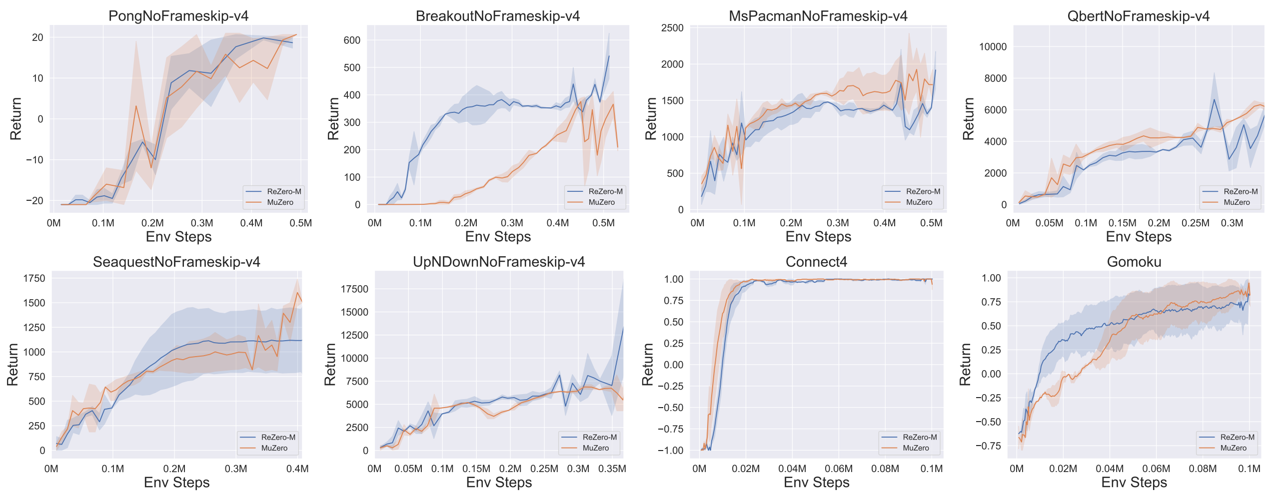

Figure 5. **Sample-efficiency** of ReZero-M vs. MuZero on six representative Atari games and two board games. The horizontal axis represents *Env Steps*, while the vertical axis indicates the *Episode Return* over 5 assessed episodes. ReZero achieves *similar* sample-efficiency than the baseline method. Mean of 5 runs; shaded areas are 95% confidence intervals.

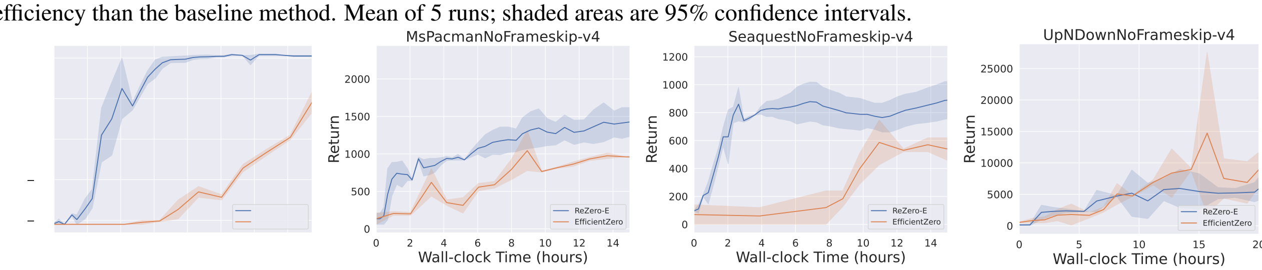

Figure 6. **Time-efficiency** of ReZero-E vs. EfficientZero on four representative Atari games. The horizontal axis represents *Wall-time* (hours), while the vertical axis indicates the *Episode Return* over 5 assessed episodes. ReZero-E achieves higher time-efficiency than the baseline method. Mean of 3 runs; shaded areas are 95% confidence intervals.

avg. wall time (h) to 100k env. steps ↓	Pong	MsPacman	Seaquest	UpNDown
ReZero-E (ours)	$\mathbf{2.3}\pm1.4$	$\mathbf{3}\pm0.3$	$\mathbf{3.1}\pm0.1$	$\mathbf{3.6}\pm0.2$
EfficientZero (Ye et al., 2021)	10 ± 0.2	12 ± 1.3	15 ± 2.3	15 ± 0.7

Table 5. **Average wall-time** of ReZero-E vs. EfficientZero on four Atari games.The time represents the average total wall-time to 100k environment steps for each algorithm. Mean and standard deviation over 5 runs. ReZero-E used $\mathbf{0.5-2}\times$ less wall-time per 100k steps than baseline methods on most games.

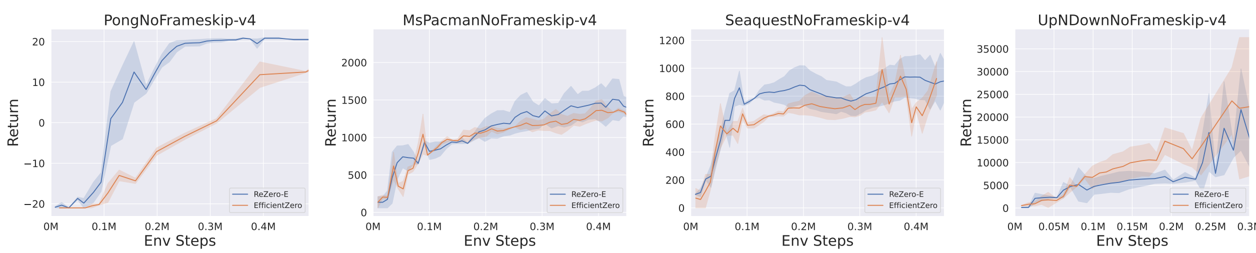

Figure 7. **Sample-efficiency** of ReZero-E vs. EfficientZero on four representative Atari games. The horizontal axis represents *Env Steps*, while the vertical axis indicates the *Episode Return* over 5 assessed episodes. ReZero-E achieves similar sample-efficiency than the baseline method. Mean of 3 runs; shaded areas are 95% confidence intervals.