

MiniZero: Comparative Analysis of AlphaZero and MuZero on Go, Othello, and Atari Games

Ti-Rong Wu, *Member, IEEE*, Hung Guei, *Member, IEEE*, Pei-Chiun Peng, Po-Wei Huang, Ting Han Wei, Chung-Chin Shih, *Member, IEEE*, and Yun-Jui Tsai

Abstract—This paper presents *MiniZero*, a zero-knowledge learning framework that supports four state-of-the-art algorithms, including AlphaZero, MuZero, Gumbel AlphaZero, and Gumbel MuZero. While these algorithms have demonstrated super-human performance in many games, it remains unclear which among them is most suitable or efficient for specific tasks. Through *MiniZero*, we systematically evaluate the performance of each algorithm in the two board games, 9x9 Go and 8x8 Othello, as well as 57 Atari games. For the two board games, using more simulations generally results in higher performance. However, the choice between AlphaZero and MuZero may differ based on game properties. For Atari games, both MuZero and Gumbel MuZero are worth considering. Since each game has unique characteristics, different algorithms and simulations yield varying results. In addition, we introduce an approach, called *progressive simulation*, which progressively increases the simulation budget during training to allocate computation more efficiently. Our empirical results demonstrate that *progressive simulation* achieves significantly superior performance in the two board games. By making our framework and trained models publicly available, this paper contributes a benchmark for future research on zero-knowledge learning algorithms, assisting researchers in algorithm selection and comparison against these zero-knowledge learning baselines. Our code and data are available at <https://rlg.iis.sinica.edu.tw/papers/minizero>.

Index Terms—AlphaZero, Atari games, deep reinforcement learning, Go, Gumbel AlphaZero, Gumbel MuZero, MuZero

I. INTRODUCTION

ALPHAGO, AlphaGo Zero, AlphaZero, and MuZero are a family of algorithms that have each made groundbreaking strides in model-based deep reinforcement learning. AlphaGo [1] was the first program to achieve super-human performance in the game of Go, demonstrated by its victory over world champion Lee Sedol in 2016. Building on this triumph, AlphaGo Zero [2] introduced the concept of *zero-knowledge learning* by removing the need for external human

knowledge. More specifically, it does not use expert game records for move prediction. Instead, AlphaGo Zero is trained solely with self-play from scratch. Experiments at the time indicate that this approach resulted in stronger play, with AlphaGo Zero beating AlphaGo 100-0. Additionally, without the need for domain-specific knowledge, self-play training could be conducted across a much wider class of games. AlphaZero [3] demonstrated that zero-knowledge learning can be extended to chess, shogi, and Go, beating state-of-the-art programs in each game. Lastly, MuZero [4] improves upon its predecessors by removing knowledge of the task environment. This includes knowledge of game rules and state transitions. By learning additional representation and dynamics models, MuZero does not have to interact with environments as it is planning ahead. This allows it to master both board games and Atari games, opening up the potential for extension to complex real-world scenarios.

While the zero-knowledge learning algorithms, AlphaZero and MuZero, have achieved super-human performance in many games, recent research highlighted that neither AlphaZero nor MuZero ensures policy improvement unless all actions are evaluated at the root of the Monte Carlo search tree [5]. This implies the potential for failure when training with limited simulations. Indeed, common settings for both algorithms use 800 Monte Carlo tree search (MCTS) [6]–[8] simulations for each move when playing board games, which can be computationally costly. Gumbel Zero algorithms, *Gumbel AlphaZero* and *Gumbel MuZero*, incorporate Gumbel noise into the original algorithms to guarantee policy improvement [5], even with fewer simulations. The two algorithms match the performance of AlphaZero and MuZero for both board games and Atari games. Moreover, Gumbel Zero demonstrates significant performance with as low as only two simulations.

Since AlphaGo's publication, many open-source projects have emerged to reproduce these algorithms for different purposes. Several projects are particularly dedicated to reproducing either the AlphaZero or MuZero algorithms for specific games. For instance, KataGo [9], LeelaZero [10], ELF OpenGo [11], and CGI [12] primarily apply the AlphaZero algorithm to the game of Go, while Leela Chess Zero [13] is designed for chess. Moreover, EfficientZero [14] introduces enhancements and implements the MuZero algorithm for Atari games.

Meanwhile, other projects aim to provide a general framework that accommodates a wide range of games. OpenSpiel [15], Polygames [16], and AlphaZero General [17] all offer frameworks based on the AlphaZero algorithm, supporting

This research is partially supported by the National Science and Technology Council (NSTC) of the Republic of China (Taiwan) under Grant NSTC 111-2222-E-001-001-MY2 and 112-2634-F-A49-004. (*Corresponding author: Ti-Rong Wu*)

Ti-Rong Wu and Hung Guei are with the Institute of Information Science, Academia Sinica, Taipei, Taiwan (e-mail: tirongwu@iis.sinica.edu.tw; hguei@iis.sinica.edu.tw).

Pei-Chiun Peng and Po-Wei Huang are with the Department of Computer Science, National Yang Ming Chiao Tung University, Hsinchu, Taiwan (e-mails: pcpeng.ee11@nycu.edu.tw; a311551048.cs11@nycu.edu.tw).

Ting Han Wei is with the School of Informatics, Kochi University of Technology, Kami City, Japan (e-mail: tinghan.wei@kochi-tech.ac.jp).

Chung-Chin Shih is with the Institute of Information Science, Academia Sinica, Taipei, Taiwan (e-mail: rockmanray@iis.sinica.edu.tw).

Yun-Jui Tsai is with the Department of Computer Science, National Yang Ming Chiao Tung University, Hsinchu, Taiwan (e-mails: b08202011.cs12@nycu.edu.tw).

various board games. In contrast, MuZero General [18] provides training across both board and Atari games based on the MuZero algorithm. The recently introduced LightZero [19] framework supports extensive zero-knowledge learning algorithms, such as AlphaZero, MuZero, and two Gumbel Zero algorithms for various environments.

Despite the variety of open-source zero-knowledge learning projects available, there is no available analysis on which algorithm – AlphaZero, MuZero, Gumbel AlphaZero, or Gumbel MuZero – is most suitable or efficient for specific tasks. To answer this question, this paper introduces *MiniZero*, a zero-knowledge learning framework that supports all four algorithms. Based on this framework, we conduct comprehensive experiments and offer detailed analyses of performances across algorithms and tasks. The evaluated games include the two board games, 9x9 Go and 8x8 Othello, as well as 57 Atari games. Furthermore, we propose a novel approach, named *progressive simulation*, which gradually increases the simulation budget during Gumbel Zero training to allocate computation more efficiently. Notably, with progressive simulation, both Go and Othello achieve higher performance compared to the original Gumbel Zero with a fixed simulation budget. We have also made our framework and all trained models publicly accessible¹, in the hopes that our empirical findings can serve as benchmarks for the community, and assist future researchers in comparing novel algorithms against these zero-knowledge learning baselines.

II. BACKGROUND

This section reviews four popular zero-knowledge learning algorithms, including the AlphaZero algorithm in Section II-A, the MuZero algorithm in Section II-B, and both the Gumbel AlphaZero and Gumbel MuZero algorithms in Section II-C. The comparison between these algorithms is summarized in Table I.

A. AlphaZero

AlphaZero [3] is a zero-knowledge learning algorithm that masters a variety of board games without using human knowledge. The network architecture consists of several residual blocks [20] and two heads, including a policy head and a value head. Given a board position, the policy head outputs a policy distribution p for possible actions, while the value head predicts an estimated outcome v . The training process comprises two components: self-play and optimization.

Self-play performs MCTS [6]–[8] for all players and every move, starting from an initial board position until the end of the game. MCTS contains three phases: selection, expansion, and backpropagation. In the selection phase, an action a is chosen for a given state s using the PUCT [2], [21] formula:

$$a^* = \underset{a}{\operatorname{argmax}} \{ Q(s, a) + c_{puct} P(s, a) \frac{\sqrt{\Sigma_b N(s, b)}}{1 + N(s, a)} \}, \quad (1)$$

where $Q(s, a)$, $P(s, a)$, and $N(s, a)$ denote the estimated value, prior probability, and visit count of a at node s , and

¹Available at: <https://github.com/rlglab/minizero/tree/d29ef42>

TABLE I
COMPARISON BETWEEN ALPHAZERO, MUZERO, GUMBEL ALPHAZERO, AND GUMBEL MUZERO

	AlphaZero	MuZero	Gumbel AlphaZero	Gumbel MuZero
Planning w/o simulator		V		V
Policy improvement			V	V
Apply to board games	V	V	V	V
Apply to Atari games		V		V

c_{puct} is an exploration hyperparameter. State s is initialized to the root of the search tree. The chosen action a is performed, upon which s is set to the resulting state. This process is repeated until s is a leaf node. Next, the leaf node is evaluated by the network during the expansion phase. All children are expanded with $Q(s, a) = N(s, a) = 0$, $P(s, a) = p_a$. Finally, during backpropagation, the estimated outcome v obtained from the value network is updated along the selection path, upwards towards the root. Complete self-play game records are stored in a replay buffer for optimization.

Data is sampled randomly from the replay buffer to optimize the network. Specifically, the policy network aims to learn p such that it matches MCTS search policy distribution π . Simultaneously, the value network is updated to minimize the error between v and the game outcome z . The optimization loss is shown in equation (2):

$$L = (z - v)^2 - \pi^\top \log p + c \|\theta\|^2, \quad (2)$$

where the last term is for regularization, in which $\|\theta\|^2$ denotes the L2 normalization of the model parameters θ , and c is a hyperparameter.

Since AlphaZero can achieve superhuman performance without the need for human knowledge, it has been widely extended to other games [15], [16] as well as non-game applications, such as optimization problems like matrix multiplication discovery [22] and sorting algorithm improvement [23].

B. MuZero

MuZero [4] builds upon the AlphaZero algorithm but incorporates neural network-based learned models to learn the environment. These models allow MuZero to plan ahead without requiring additional environment interactions, as summarized in Table I. This is especially useful when environment simulators are not easily accessible or costly. MuZero not only matches the super-human performance of AlphaZero in board games such as chess, shogi, and Go, it also achieves new state-of-the-art performance in Atari games.

MuZero employs three networks: *representation*, *dynamics*, and *prediction*. The prediction network, denoted by $f(s) = (p, v)$, is similar to the two-headed network used in AlphaZero. The representation and dynamics networks play an important role in learning abstract hidden states that represent the real environment. First, the representation network, denoted by $h(o) = s$, converts environment observations o into a hidden state s . Next, the dynamics network, denoted by $g(s, a) = (r, s')$, learns environment transitions. For a given hidden state

s and action a , it generates the next hidden state s' and the state-action pair's associated reward r . Essentially, the dynamics network serves as a surrogate environment simulator, allowing MuZero to simulate trajectories during planning without having to sample actions.

The training process for MuZero is the same as AlphaZero, which includes both self-play and optimization. In self-play, MCTS is executed for each move, and the completed games are stored in a replay buffer. However, different from AlphaZero, the MCTS in MuZero first converts the environment observation into a hidden state, then conducts planning through the dynamics network. During optimization, MuZero unrolls K steps, ensuring that the model is in alignment with sequences sampled from the replay buffer. In addition, the loss function is modified:

$$l = \sum_{k=0}^K l^p(\pi_k, p_k) + \sum_{k=0}^K l^v(z_k, v_k) + \sum_{k=0}^K l^r(u_k, r_k) + c\|\theta\|^2, \quad (3)$$

where l^p , l^v , and l^r denote the policy, value, and reward losses respectively. The policy and value losses are similar to AlphaZero. Note that z is calculated based on the n -step return in Atari games. The reward loss minimizes the error between the predicted immediate reward r and the observed immediate reward obtained from the environment u .

There are several enhancements for MuZero. Sampled MuZero [24] adapts MuZero for environments with continuous actions, while Stochastic MuZero [25] modifies MuZero to support stochastic environments. Additionally, both EfficientZero [14] and Gumbel MuZero [5] introduce different methods to improve the learning efficiency.

C. Gumbel AlphaZero and Gumbel MuZero

The Gumbel Zero algorithms [5] were proposed to guarantee policy improvement, as summarized in Table I. This improves training performance with smaller simulation budgets. The modifications include MCTS root node action selection, environment action selection, and policy network update.

For MCTS root node action selection, Gumbel Zero algorithms employ the Gumbel-Top- k trick [26] and the sequential halving algorithm [27] to select actions at the root node. The Gumbel-Top- k trick samples the top k actions with higher $G(a) + \logits(a)$, where $G(a)$ is the sampled Gumbel variable and $\logits(a)$ is the unnormalized prediction generated by the policy network. If the simulation count for the MCTS, n , is set to k , k actions are sampled, using up the simulation budget. The final chosen action is the one with the highest $G(a) + \logits(a) + \sigma(q(a))$, where $q(a)$ is the Q value of a and σ is a monotonically increasing transformation. On the other hand, if n is set to larger than k , sequential halving is applied to divide the search budget into several phases. This allocates more simulations to better actions. In the first phase, the search starts with k sampled actions. After each phase, the actions are sorted according to $G(a) + \logits(a) + \sigma(q(a))$, where the actions in the bottom half are discarded from consideration. In the last phase, only one action is retained. This remaining action is the most visited action and the one that is chosen.

For the environment action selection, we explicitly select the best action from the root node utilizing the aforementioned method. This is in contrast to the non-Gumbel algorithms, where the action is sampled according to the search policy – a distribution that is formed by the visit counts of the root actions.

For policy network updates, due to the very few actions sampled, we cannot use the search visit count distribution as the policy network training target. The Gumbel trick uses Q values to derive a policy training target. For visited children, we use the sampled Q value. For those that have not been visited, we use the value network instead.

III. MINIZERO

This section first presents the *MiniZero* framework in Section III-A. Then, section III-B introduces an estimated Q value method for non-visited actions. Finally, section III-C proposes a new training approach for Gumbel Zero algorithms.

A. Framework Design

The architecture of *MiniZero* shown in Fig. 1 comprises four components, including a *server*, one or more *self-play workers*, an *optimization worker*, and *data storage*. We describe each component in the next paragraph.

The server is the core component in *MiniZero*, controlling the training process and managing both the self-play and optimization workers. The training process contains several iterations. For each iteration, the server first instructs all self-play workers to generate self-play games simultaneously by using the latest network. Each self-play worker maintains multiple MCTS instances to play multiple games simultaneously with batch GPU inferencing to improve efficiency. Specifically, the self-play worker runs the selection for each MCTS to collect a batch of leaf nodes and then evaluates them through batch GPU inferencing. Finished self-play games are sent to the server and forwarded to the data storage by the server. Once the server accumulates the necessary self-play games, it then stops the self-play workers and instructs the optimization worker to start network updates. The optimization worker updates the network over steps using data sampled from the replay buffer. Generally, the number of optimized steps is proportional to the number of collected self-play games to prevent overfitting. The optimization worker stores updated networks into the data storage. The server then starts the next iteration. This process continues until the training reaches a predetermined iteration I .

The server and workers communicate using TCP connections for message exchange. The data storage uses the Network File System (NFS) for sharing data across different machines. This is an implementation choice; a simpler file system can suffice if distributed computing is not employed.

MiniZero supports various zero-knowledge learning algorithms, including AlphaZero, MuZero, Gumbel AlphaZero, and Gumbel MuZero. To ensure execution efficiency, our implementation utilizes C++ and PyTorch [28]. To support experiments on Atari games, it also utilizes the Arcade Learning Environment (ALE) [29], [30].

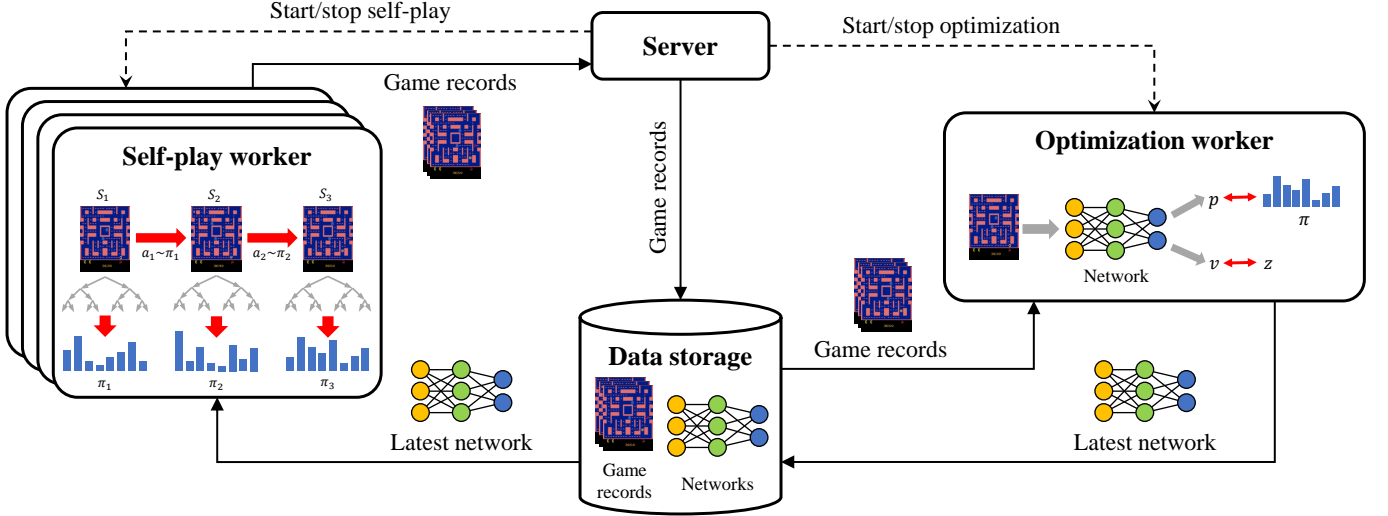


Fig. 1. The MiniZero architecture, includes four components: a server, self-play workers, an optimization worker, and data storage.

B. MCTS Estimated Q Value for Non-Visited Actions

While our implementation mainly follows the original design, we introduce a modification to enhance exploration. The method is similar to that used in ELF OpenGo [11] and EfficientZero [14], in which all non-visited nodes are initialized to an estimated Q value instead of 0 during MCTS.

To elaborate, $\hat{Q}(s)$ is introduced as an estimated Q value when $N(s, a) = 0$:

$$Q(s, a) = \begin{cases} Q(s, a) & N(s, a) > 0 \\ \hat{Q}(s) & N(s, a) = 0. \end{cases} \quad (4)$$

The value of $\hat{Q}(s)$ is determined by two statistics $N_\Sigma(s)$ and Q_Σ from MCTS:

$$\begin{aligned} N_\Sigma(s) &= \sum_b \mathbf{1}_{N(s,b)>0} \\ Q_\Sigma(s) &= \sum_b \mathbf{1}_{N(s,b)>0} Q(s, b), \end{aligned} \quad (5)$$

where $\mathbf{1}_{N(s,b)>0}$ is the characteristic function that filters the visited actions b at s , $N_\Sigma(s)$ represents the number of children that have been visited, and Q_Σ denotes the value sum of these visited children. We use different calculations for $\hat{Q}(s)$ in board games than Atari games. To avoid performing forced exploration (as in breadth-first search), we bias the estimated Q to a smaller value by virtually sampling the action once with a losing outcome:

$$\hat{Q}(s) = \frac{Q_\Sigma(s)}{N_\Sigma(s) + 1}. \quad (6)$$

In contrast, we encourage enhanced exploration in Atari games:

$$\hat{Q}(s) = \begin{cases} \frac{Q_\Sigma(s)}{N_\Sigma(s)} & N_\Sigma(s) > 0 \\ 1 & N_\Sigma(s) = 0. \end{cases} \quad (7)$$

C. Progressive Simulation for Gumbel Zero

Gumbel Zero algorithms can guarantee policy improvement and achieve comparable performance using only a few

simulations. However, more simulations can still improve performance when planning with Gumbel, especially if the lookahead depth is relatively deeper. To take advantage of this effect, we propose a method called *progressive simulation*. We wish to weight the number of simulations so that it gradually increases during Gumbel Zero training, all while ensuring no additional computing resources are consumed.

Given a total number of iterations I and the simulation budget B , the simulation count n_i for each iteration is computed as $\frac{B}{I}$ in the unaltered Gumbel Zero training process. Namely, the simulation count remains consistent throughout training. With progressive simulation, we introduce two hyperparameters, N_{min} and N_{max} . The training uses the same simulation budget B by adjusting the simulation count n_i dynamically during different iterations, where $n_i \in [N_{min}, N_{max}]$.

Algorithm 1 allocates the number of simulations n_i for each iteration such that it satisfies $B = \sum n_i$ as follows. First, all n_i are initialized to N_{min} at the beginning, as in line 2. Then, we progressively set a target simulation of N_{max} , $\frac{N_{max}}{2}$, $\frac{N_{max}}{4}$, and so on to remaining unallocated iterations, as in lines 3-10. For each target simulation, up to half of the remaining budget is allocated. Finally, when the remaining budget is not enough for a target or all n_i have been allocated once, the remaining budget is allocated to n_i with the lowest simulations, as in lines 11-16.

Take $I = 300$, $B = 4,800$, and $(N_{min}, N_{max}) = (2, 200)$ as an example. First, n_1 to n_{300} are initialized to $N_{min} = 2$ simulations, with a remaining budget of 4,200 simulations. Next, we progressively allocate the target number of simulations, starting from 200. For the target of 200 simulations, we need to allocate an additional 198 simulations per iteration (two simulations are already allocated, so $200 - 2 = 198$). Half of the remaining budget, 2,100 simulations, is split among $\lfloor \frac{2,100}{198} \rfloor = 10$ iterations. Therefore, n_{291} to n_{300} are allocated an additional 198×10 simulations. Next, for the target of 100 simulations (with a remaining budget of $4,200 - 1,980 = 2,220$), n_{280} to n_{290} are allocated. The procedure repeats until the target of 3 simulations is allocated

Algorithm 1 Progressive Simulation Budget Allocation

Require: total iterations I
Require: simulation budget B
Require: simulation boundary (N_{min}, N_{max})

- 1: $N \leftarrow []$ \triangleright simulation budget list for I iterations
- 2: $B \leftarrow B - I \times N_{min}$ \triangleright allocate N_{min} to each iteration
- 3: $n \leftarrow N_{max}$
- 4: **while** $B > 0$ **and** $n > N_{min}$ **and** $I - |N| > 0$ **do**
- 5: $i \leftarrow \min(\lfloor \frac{B \times 0.5}{n - N_{min}} \rfloor, I - |N|)$
- 6: insert i elements of n at the front of N
- 7: $B \leftarrow B - i \times (n - N_{min})$
- 8: $n \leftarrow \lfloor \frac{n}{2} \rfloor$
- 9: **end while**
- 10: insert $(I - |N|)$ elements of N_{min} at the front of N
- 11: **while** $B > 0$ **do**
- 12: $k \leftarrow$ count the number of occurrences of $\min(N)$ in N
- 13: $i \leftarrow \min(B, k)$
- 14: add 1 to each of N from $(k - i + 1)$ th through k th
- 15: $B \leftarrow B - i$
- 16: **end while**
- 17: **return** N \triangleright allocation results s.t. $\text{sum}(N) = B$

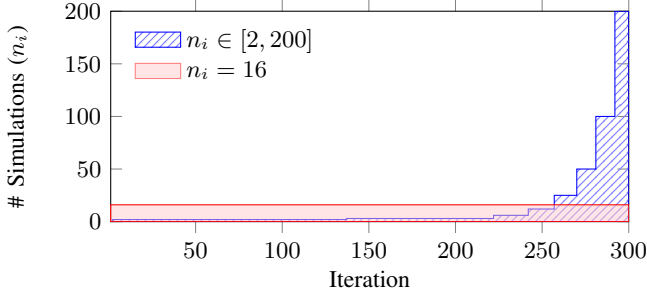


Fig. 2. Simulation budget allocation for progressive simulation with a setting of $I = 300$, $B = 4800$, and $(N_{min}, N_{max}) = (2, 200)$. $n_i \in [2, 200]$ depicts the simulations of each iteration for the progressive simulation; $n_i = 16$ depicts those for the baseline that uses a fixed simulation of 16. Note that the baseline has the same budget as the progressive simulation.

to n_{179} to n_{220} . The number of simulations n_1 to n_{178} remain at $N_{min} = 2$ simulations, the same as its initialized value. Finally, the remaining 43 simulations are assigned to n_{136} to n_{178} , resulting in the allocation as shown in Fig. 2.

IV. EXPERIMENTS

We evaluate the performance of four zero-knowledge learning algorithms, AlphaZero, MuZero, Gumbel AlphaZero, and Gumbel MuZero. For simplicity, the four algorithms are denoted as α_0 , μ_0 , $g\text{-}\alpha_0$, and $g\text{-}\mu_0$, respectively. This section is organized as follows. First, Section IV-A introduces the training procedure and settings. Second, sections IV-B, IV-C, and IV-D compare the performance differences when using different algorithms on the two board games and Atari games, respectively. Third, section IV-E conducts an ablation study to compare the performance of the estimated Q value. Finally, section IV-F analyzes the proposed progressive simulation method.

TABLE II
HYPER-PARAMETERS FOR BOARD GAMES AND ATARI GAMES

Parameter	Board Games	Atari Games
Iteration		300
Optimizer		SGD
Optimizer: learning rate		0.1
Optimizer: momentum		0.9
Optimizer: weight decay		0.0001
Training steps		60K
# Blocks	3	2
Batch size	1024	512
Replay buffer size	40K games	1M frames
Max frames per episode	-	108K
Discount factor	-	0.997
Priority exponent (α)	-	1
Priority correction (β)	-	0.4
Bootstrap step (n-step return)	-	5

A. Setup

All experiments are conducted on one machine equipped with two Intel Xeon E5-2678 v3 CPUs, and four GTX 1080Ti GPUs. We apply α_0 and $g\text{-}\alpha_0$ to the two board games, 9x9 Go and 8x8 Othello, and use μ_0 and $g\text{-}\mu_0$ for both board games and 57 Atari games. Table II lists the hyperparameters. Generally, we follow the same hyperparameters and network architectures as the original AlphaZero, MuZero, and Gumbel Zero paper but with some differences described as follows.

For board games, we train two models for α_0 and μ_0 with 200 MCTS simulations. For $g\text{-}\alpha_0$ and $g\text{-}\mu_0$, each algorithm is trained with two models with 2 and 16 MCTS simulations. The numbers of simulations are chosen to follow the settings proposed in [5]. We omit training both $g\text{-}\alpha_0$ and $g\text{-}\mu_0$ with 200 simulations, as the experiments in [5] demonstrate that Gumbel Zero matches the performance of both AlphaZero and MuZero when training with the same number of simulations. All six models use the same network architecture containing 3 residual blocks. Note that in our MuZero network design, both the representation and dynamics networks contain 3 residual blocks. During training, for each iteration, self-play workers generate 2,000 games and the optimization worker updates the network with 200 steps by using randomly sampled data from the most recent 40,000 games in the replay buffer. For example, with a total of 300 iterations, each model is trained with a total of 600,000 self-play games and optimized for 60,000 training steps with a batch size of 1,024.

For Atari games, we train one μ_0 model with 50 MCTS simulations, and two $g\text{-}\mu_0$ models, one with 2 and the other with 18 simulations. The choices of simulations follow the original setup in [5]. The network block size is reduced to 2 residual blocks. For each iteration, the self-play workers generate 250 intermediate sequences, where each contain 200 moves. The optimization worker updates the network with 200 steps by using data randomly sampled from the most recent 1 million frames. We calculate the n-step return by bootstrapping with five steps. Overall, each model is trained for a total of 600,000 intermediate sequences and optimized for 60,000 training steps with a batch size of 512.

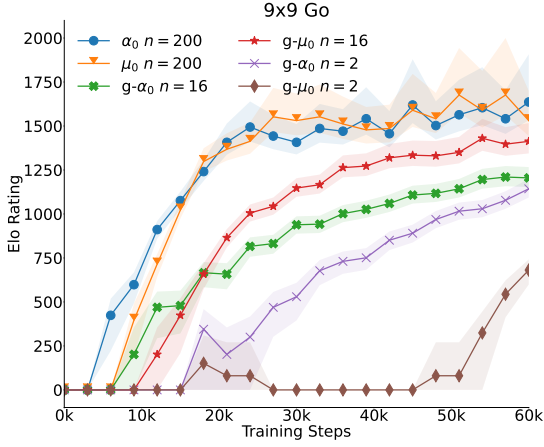


Fig. 3. Evaluation of different zero-knowledge learning settings in 9x9 Go. The x-axis represents the number of neural network training steps, while the y-axis represents Elo ratings. The shaded area is the error bar with 95% confidence interval.

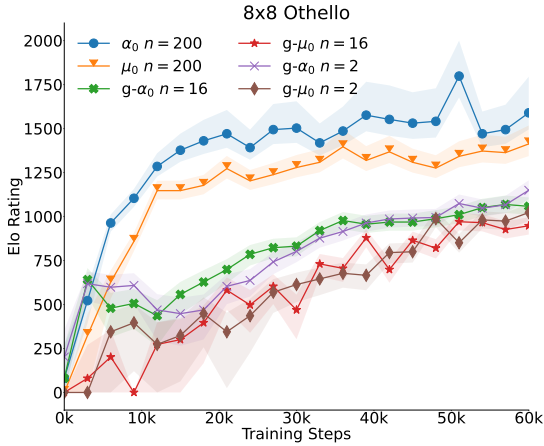


Fig. 4. Evaluation of different zero-knowledge learning settings in 8x8 Othello. The x-axis represents the number of neural network training steps, while the y-axis represents Elo ratings. The shaded area is the error bar with 95% confidence interval.

B. Comparison of Different Algorithms on Board Games

We compare the performance of six different zero-knowledge learning models, including α_0 $n = 200$, μ_0 $n = 200$, $g\text{-}\alpha_0$ $n = 2$, $g\text{-}\alpha_0$ $n = 16$, $g\text{-}\mu_0$ $n = 2$, and $g\text{-}\mu_0$ $n = 16$, on 9x9 Go and 8x8 Othello. Each model is evaluated against a specific baseline model with Elo ratings [3], which is chosen to be neither excessively strong nor overly weak, for better discrimination. Specifically, we choose μ_0 $n = 200$ trained for 15,000 training steps (75 iterations) for 9x9 Go and 10,000 training steps (50 iterations) for 8x8 Othello as the baselines. Fig. 3 and Fig. 4 show the strength comparison between models for the two games. For each curve, we sample the networks every 3,000 training steps (15 iterations) and evaluate 200 games against the baseline model. Namely, a total of 21 network checkpoints (including the initial network) are evaluated for each model. All models use 400 simulations per turn during evaluation, regardless of the simulation count used during training.

In 9x9 Go, training with more simulations generally yields

stronger results than training with fewer simulations. For $n = 200$, both α_0 and μ_0 perform similarly, surpassing the performance of the four other Gumbel Zero settings. Next, for $n = 16$, $g\text{-}\mu_0$ outperforms $g\text{-}\alpha_0$, indicating that the MuZero algorithm might be more suitable for the game of Go. For $n = 2$, the learning curve of $g\text{-}\alpha_0$ grows slowly but eventually reaches a similar performance to $g\text{-}\alpha_0$ $n = 16$. However, $g\text{-}\mu_0$ $n = 2$ demonstrates the poorest performance among the six models. The reason could be that only two simulations are not sufficient to learn the environment representation and dynamics in 9x9 Go for $g\text{-}\mu_0$. In conclusion, for 9x9 Go, both AlphaZero and MuZero are appropriate. However, without a sufficient number of simulations, MuZero might not perform well.

Next, in 8x8 Othello, α_0 and μ_0 still outperform $g\text{-}\alpha_0$ and $g\text{-}\mu_0$, demonstrating that using more simulations directly affects the performance. Different from 9x9 Go, α_0 performs better than μ_0 with a significant Elo rating difference. This phenomenon also exists in Gumbel Zero where $g\text{-}\alpha_0$ is generally better than $g\text{-}\mu_0$. A possible explanation is that Othello has a more dramatic change on the board with each turn due to the frequent piece flipping across the board. The dynamics network might therefore find it more difficult to learn state transitions. Interestingly, in Gumbel Zero, we observe that the performance of $n = 2$ is equal to or might even be slightly better than $n = 16$ for both $g\text{-}\alpha_0$ and $g\text{-}\mu_0$. This contradicts the results in Go, where more simulations usually yield higher Elo ratings. This discrepancy might be attributed to the fact that Othello often has fewer legal moves relative to Go, so that fewer simulations might be sufficient in exploring optimal moves and avoiding bad ones.

In summary, given sufficient computing resources, the choice of α_0 or μ_0 with more simulations, such as $n = 200$, is suitable. However, with limited computing resources, we can consider using $g\text{-}\alpha_0$ or $g\text{-}\mu_0$ with fewer simulations. The choice of $g\text{-}\alpha_0$ and $g\text{-}\mu_0$ with different simulation settings might depend on the property of the game. For games with more dramatic environment changes like 8x8 Othello, $g\text{-}\alpha_0$ is a better choice than $g\text{-}\mu_0$ because it can interact with the actual environment during planning, without relying on learned environments. In terms of the choice of simulation count in Gumbel Zero, for games that generally have many large branching factors like 9x9 Go, a higher number of simulations is essential for both $g\text{-}\alpha_0$ and $g\text{-}\mu_0$.

C. Comparison of Different Algorithms on Board Games under the Same Training Time

The previous subsection mainly compares the performance of algorithms with the same number of training steps (iterations). However, it is an interesting question whether using $n = 2$ and $n = 16$ can attain similar performance to $n = 200$ under the same training time. To investigate this issue, we select the game of 8x8 Othello and train $g\text{-}\alpha_0$ $n = 2$ and $g\text{-}\alpha_0$ $n = 16$ using the same amount of time as α_0 $n = 200$. Ordinarily, for every 200 training steps (one iteration of training), α_0 $n = 200$, $g\text{-}\alpha_0$ $n = 16$, and $g\text{-}\alpha_0$ $n = 2$, take around 378.51 seconds, 40.62 seconds,

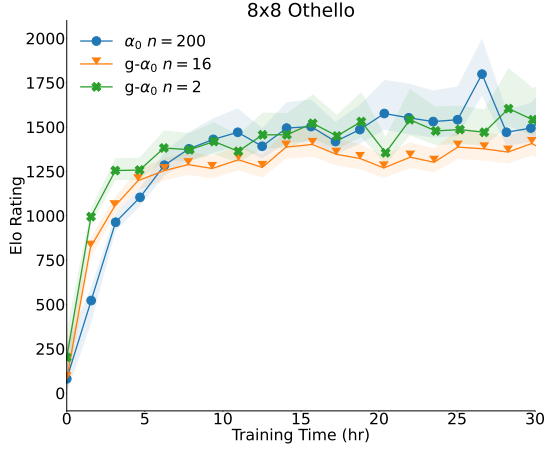


Fig. 5. Comparing the playing performance of α_0 and $g\text{-}\alpha_0$ on different simulations under the same amount of training time in 8x8 Othello.

and 23.34 seconds. We set the training process for all three models to roughly the same amount of time, for a total of 60,000, 560,000, and 972,000 training steps, i.e. 300, 2,800, and 4,860 iterations, respectively. The result is shown in Fig. 5. During the initial five hours of training, both $g\text{-}\alpha_0$ $n = 2$ and $n = 16$ perform slightly better than α_0 $n = 200$. After training for the same amount of time, $g\text{-}\alpha_0$ $n = 2$ and α_0 $n = 200$ achieve similar playing strength. However, $g\text{-}\alpha_0$ $n = 16$ performs slightly worse than others, consistent with previous experiments described in section IV-B.

D. Comparison of Different Algorithms on Atari Games

For Atari games, we use a similar evaluation approach as Muesli [31], where the average returns are calculated based on the last 100 finished training episodes from self-play games. Table III lists the average returns of 57 Atari games, while Fig. 12 depicts the learning curves for each game. Among 57 Atari games, we observed that μ_0 $n = 50$, $g\text{-}\mu_0$ $n = 18$, and $g\text{-}\mu_0$ $n = 2$ perform the best in 40, 10, and 13 games, respectively. Generally, μ_0 $n = 50$ consistently outperforms $g\text{-}\mu_0$ $n = 18$, and $g\text{-}\mu_0$ $n = 18$ outperforms $g\text{-}\mu_0$ $n = 2$.

There are three cases worth discussing the performance of $g\text{-}\mu_0$ versus μ_0 . First, we find that μ_0 $n = 50$ achieves noticeably higher average returns than $g\text{-}\mu_0$ $n = 18$ and $g\text{-}\mu_0$ $n = 2$ in some games². For example, in *centipede*, the average returns of μ_0 $n = 50$ achieves 49,823.07, while the $g\text{-}\mu_0$ $n = 18$ and $g\text{-}\mu_0$ $n = 2$ only reach 30,749.87 and 9,946.83, respectively. We suspect that these games require meticulous planning and precise action sequences. Therefore, having more simulations can be critical for achieving higher returns. In $g\text{-}\mu_0$ $n = 2$, only two actions are evaluated, which makes exploring better actions more challenging without deeper planning.

Second, we discuss the case for $g\text{-}\mu_0$ outperforming μ_0 . Specifically, for the three games, *gravitar*, *phoenix*, and *video_pinball*, $g\text{-}\mu_0$ $n = 2$ outperformed significantly better than μ_0 $n = 50$. To investigate which of the two settings, the Gumbel trick or the number of simulations, contribute more

to this phenomenon, we additionally trained $g\text{-}\mu_0$ $n = 50$ for these three games, as shown in Fig. 6. For the two games, *gravitar* and *phoenix*, performance does not improve even with 50 simulations, regardless of whether the Gumbel trick is used. We observe that *gravitar*³ has highly unpredictable environment changes that are hard to learn, making it difficult to learn an accurate dynamics network. As the tree digs deeper, the simulated environment diverges more from the real environment, leading to increasingly inaccurate policy and value estimates. Next, we investigate the game of *phoenix*, which is a shooting game. We notice that the boss (an alien pilot) constantly fires bullets towards the center of the game, making it more challenging for the player to survive when close to the boss. Surprisingly, μ_0 $n = 50$ tends to stay at the leftmost place, which is outside the firing range of the boss, to avoid death, as shown in Fig. 7a. With deep planning using 50 simulations, μ_0 $n = 50$ sufficiently explores to understand that approaching the boss too closely results in death easily. This results in a longer survival time, but the returns do not increase without beating the boss, leading to lower average returns. In contrast, due to a smaller number of two simulations and the Gumbel noise, $g\text{-}\mu_0$ $n = 2$ inevitably chooses actions that bring it closer to the boss, as shown in Fig. 7b. Although this results in a shorter survival time for $g\text{-}\mu_0$ $n = 2$, it allows the agent to explore another strategy to defeat the boss and obtain higher returns. For the game of *video_pinball*, $g\text{-}\mu_0$ $n = 50$ has nearly the same performance as $g\text{-}\mu_0$ $n = 2$, suggesting that the Gumbel trick has a positive impact on its performance.

Finally, although $g\text{-}\mu_0$ uses fewer simulations than μ_0 , there are some games where $g\text{-}\mu_0$ $n = 2$ and $g\text{-}\mu_0$ $n = 18$ achieve similar returns to those of μ_0 $n = 50$. For example, in *boxing* and *qbert*, the game environment does not have extensive changes and offers straightforward scoring opportunities. Hence, the agent can learn the environment transitions easily while also not requiring intricate planning. Consequently, using the Gumbel approach allows $g\text{-}\mu_0$ to attain almost identical training curves as μ_0 .

It is also worth mentioning that several games⁴ achieve nearly the same performance as the original MuZero paper even if the models are trained with a smaller network architecture (2 blocks compared to 16 blocks) and fewer environment frames (15 million frames compared to 20 billion frames).

Therefore, these games might serve as exemplary benchmarks to verify the zero-knowledge learning algorithms without requiring a large amount of computational resources.

In summary, our experiments show that the performance of μ_0 and $g\text{-}\mu_0$ varies across different Atari games. This indicates that each Atari game has unique characteristics, and using different zero-knowledge learning algorithms and simulations can yield varying results. We present three cases to illustrate that the choice between $g\text{-}\mu_0$ and μ_0 , and the number of simulations might depend on the complexity of the game, the predictability of the environment, and the

²*breakout*, *centipede*, *crazy_climber*, *enduro*, *fishing_derby*, *kanaroo*, *kung_fu_master*, *private_eye*, *up_n_down*.

³*gravitar* is a game that controls a spaceship with various missions on different planets. The mission complexity will become progressively more difficult and the environment will change significantly when the mission is completed.

⁴*boxing*, *freeway*, *hero*, *pong*, *private_eye*, *enduro*, and *solaris*.

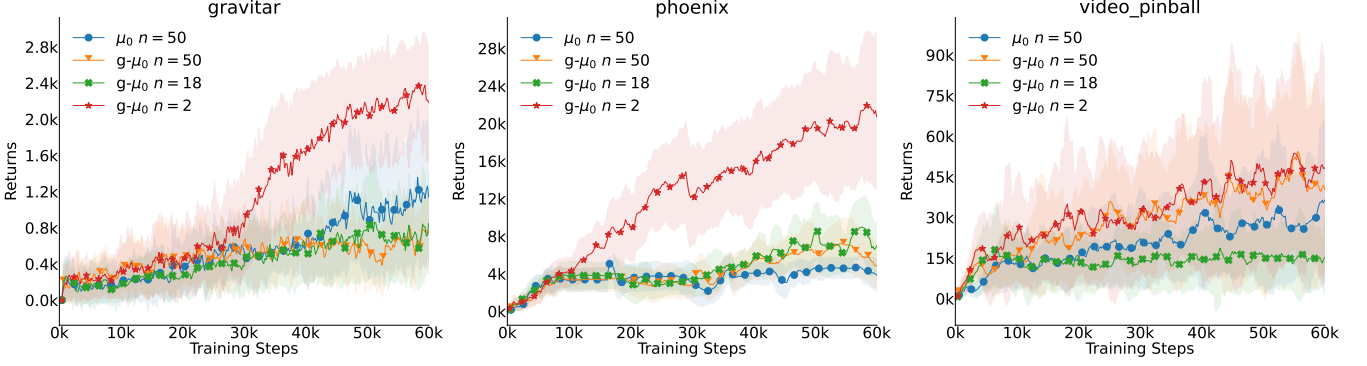


Fig. 6. The $g\text{-}\mu_0$ $n = 50$ experiment on three Atari games, including *gravitar*, *phoenix*, and *video_pinball*.

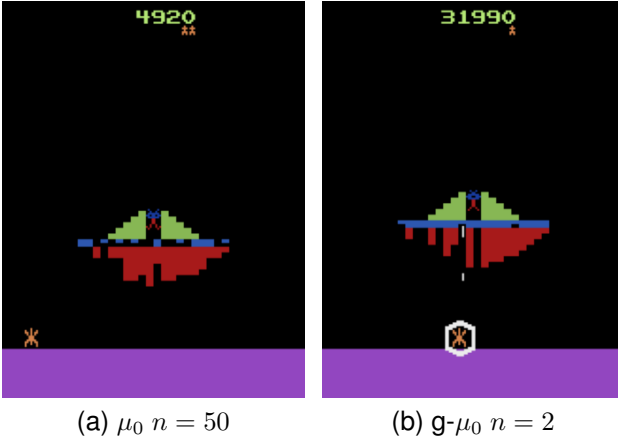


Fig. 7. A screenshot from the final iteration of self-play games in *phoenix*: (a) μ_0 $n = 50$ tends to stay at the leftmost position to avoid fire from the boss (alien pilot), and (b) $g\text{-}\mu_0$ $n = 2$ tries to move to the center and target the boss.

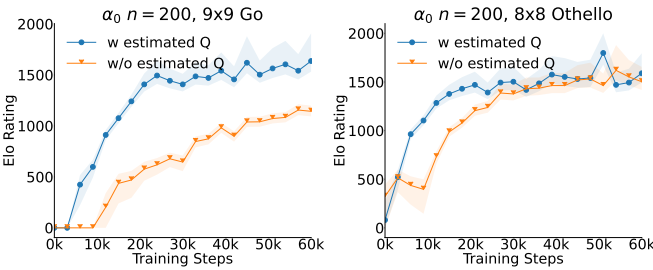


Fig. 8. Ablation study of whether using estimated Q value for non-visited actions for α_0 $n = 200$ on board games.

benefits of planning for the game. These results contribute to a deeper understanding of zero-knowledge learning algorithms. We expect that our findings will provide valuable insights for future research studying Atari games with these zero-knowledge learning algorithms.

E. Ablation Study on Estimated Q value

We conduct an ablation study to determine the impact of using the estimated Q value in MCTS, as described in section III-B. Specifically, we compare the performance using

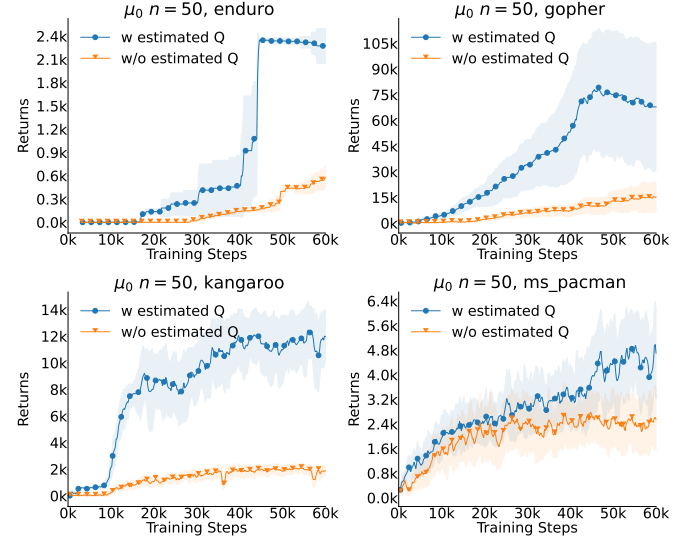


Fig. 9. Ablation study of whether using estimated Q value for non-visited actions for μ_0 $n = 50$ on four Atari games: *enduro*, *gopher*, *kangaroo*, and *ms_pacman*.

estimated Q value with α_0 for 9x9 Go and 8x8 Othello, and with μ_0 for four Atari games: *enduro*, *gopher*, *kangaroo*, and *ms_pacman*. The training settings follow those introduced above.

Fig. 8 and Fig. 9 illustrate the training results in either Elo ratings for board games or average returns for Atari games. As shown in these figures, using the estimated Q value significantly outperforms the method without the estimated Q value, with the exception of 8x8 Othello. Furthermore, the learning curves using the estimated Q value consistently outperform those without throughout the training. This indicates that for planning problems with sufficiently large action spaces, more exploration can be more beneficial.

F. Progressive Simulation in Gumbel Zero

We examine the performance of Gumbel Zero trained with *progressive simulation*, as introduced in section III-C. For board games, we train $g\text{-}\alpha_0$ and $g\text{-}\mu_0$ with the simulation boundary $(N_{min}, N_{max}) = (2, 200)$, where the simulation

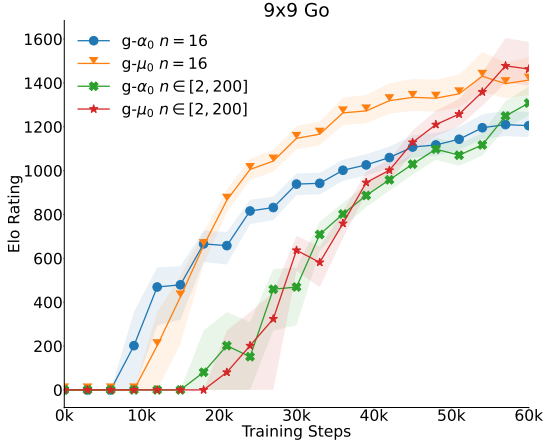


Fig. 10. Progressive simulation for Gumbel Zero in 9x9 Go games.

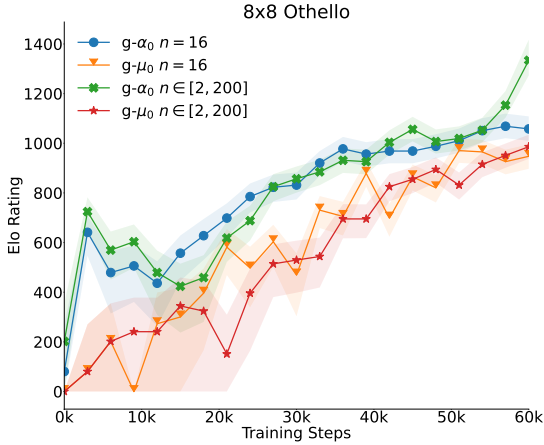


Fig. 11. Progressive simulation for Gumbel Zero in 8x8 Othello games.

budget is the same as $n = 16$. Note that under this setting, the simulation budget allocation of each iteration is the same as in Fig. 2. The evaluation results for both 9x9 Go and 8x8 Othello are shown in Fig. 10 and Fig. 11. In general, the models trained with progressive simulation outperform those trained without by the end of training. However, we observe that both $g-\alpha_0$ and $g-\mu_0$ with progressive simulation are weaker than those without before 54,000 training steps (270 iterations) for both games. Most notably, in the case of 8x8 Othello, $g-\alpha_0 n \in [2, 200]$ jumps significantly in Elo rating – by 276 – between 50,000 to 60,000 training steps. More specifically, the simulation count is increased to over $n = 16$ after 51,000 training steps, as shown in Fig. 2.

These observations corroborate existing findings on combining machine learning with planning. It has been shown that when evaluations – both value and policy – are inaccurate, planning can actually lead to worse decisions [32]. In the early training stages of AlphaZero and MuZero, the agent’s evaluations are certainly very inaccurate. The progressive simulation method addresses this by conserving the simulation budget for later stages of training, where the agent may benefit the most from planning with much more accurate evaluations.

Next, we evaluate the progressive simulation method on 57 Atari games by training $g-\mu_0$ with a simulation boundary

$(N_{min}, N_{max}) = (2, 50)$, where the simulation budget is the same as $n = 18$. The results are shown in Table III and Fig. 12. From the table, we find that progressive simulation achieves lower human normalized mean returns (359.97%) compared to the baseline model $g-\mu_0 n = 18$ (395.87%), which uses the same simulation budget. Overall, progressive simulation only outperforms $g-\mu_0 n = 18$ in 25 games among 57 games.

For cases that use progressive simulation significantly outperforms $g-\mu_0 n = 18$, such as *battle_zone*, *kangaroo*, and *jamesbond*. The average returns of these three games gradually increase during training. This situation is similar to that of board games, where fewer simulations are allocated during the early stages of training. Then, simulations are gradually increased to perform meticulous planning with higher simulation counts, obtaining better scores.

However, not all Atari games exhibit the same characteristics as board games in terms of increasing simulations. For games such as *gravitar*, training with fewer simulations outperforms training with more simulations. Thus, using progressive simulation causes training to become less efficient as it progresses. Specifically, it performs well in the early training stages but converges when the number of simulations becomes excessive. For games such as *asterix*, *private_eye*, and *surround*, the opposite is that training requires sufficient simulations. Namely, the use of progressive simulation results in wasted early stages of training.

Furthermore, we observe that several games⁵ experience a significant decline in score when the simulation counts are changed. Despite this decline, it usually recovers after several training steps. We hypothesize that the agent might settle into a sub-optimal strategy when training with a fixed simulation. As the number of simulations increases, the strategy shifts into different playing styles, which may impact the performance both positively and negatively, before long-term improvements. Therefore, the model may require a significantly larger number of training steps to accommodate these shifts and subsequent adjustments.

In conclusion, the ablation study on the progressive simulation method suggests that adjusting the simulation count during training does not necessarily lead to improvement in Atari games. Since each Atari game has distinct characteristics, the results vary, as demonstrated in section IV-D. A universally beneficial method for training Atari games is yet to be discovered.

V. CONCLUSION

This paper introduces *MiniZero*, a zero-knowledge learning framework that supports four algorithms, including AlphaZero, MuZero, Gumbel AlphaZero, and Gumbel MuZero. We compare the performance of these algorithms with different simulations on 9x9 Go, 8x8 Othello, and 57 Atari games.

For board games, given the same amount of training data, using more simulations generally results in higher performance for both AlphaZero and MuZero algorithms. Nevertheless, when trained for an equivalent amount of time (computing resources), Gumbel Zero with fewer simulations can achieve

⁵*amidar*, *berzerk*, *breakout*, *centipede*, *crazy_climber*, *qbert*, and *robotank*.

performance nearly on par with AlphaZero/MuZero using more simulations. We also propose an efficient approach, named *progressive simulation*, to reallocate the simulation budget under limited computing resources. This approach begins with fewer simulations, which is gradually increased throughout training. Our experiments demonstrate that models trained with progressive simulation outperform those trained without it. Furthermore, the choice between AlphaZero and MuZero may differ according to specific game properties. Our experiments indicate that MuZero excels in 9x9 Go, whereas AlphaZero is superior in 8x8 Othello.

The results for Atari games are different from board games. While training with 50 simulations generally yields better results than 18 and 2 simulations in terms of human normalized mean returns, it is noteworthy that using 50 simulations does not consistently outperform 18 and 2 simulations in every game. Our experiments suggest that the performance of different simulation settings is highly correlated to distinct game characteristics. Therefore, the progressive simulation method may not be necessarily optimal for all cases.

Currently, our framework only supports four fundamental zero-knowledge learning algorithms. The framework can still be improved with several extensions, such as incorporating Reanalyze techniques [4], providing continuous action spaces [25], supporting stochastic environments [24], and including more games. Besides, we can design different progressive simulation strategies, such as adjusting simulations during training without a pre-determined total budget. In conclusion, our framework, along with all trained models, is publicly available online. We expect that these trained models can serve as benchmarks for the game community in the future.

REFERENCES

- [1] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, “Mastering the game of Go with deep neural networks and tree search,” *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016.
- [2] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, “Mastering the game of Go without human knowledge,” *Nature*, vol. 550, no. 7676, pp. 354–359, Oct. 2017.
- [3] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, “A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play,” *Science*, vol. 362, no. 6419, pp. 1140–1144, Dec. 2018.
- [4] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, T. Lillicrap, and D. Silver, “Mastering Atari, Go, chess and shogi by planning with a learned model,” *Nature*, vol. 588, no. 7839, pp. 604–609, Dec. 2020.
- [5] I. Danihelka, A. Guez, J. Schrittwieser, and D. Silver, “Policy improvement by planning with Gumbel,” in *International Conference on Learning Representations*, Apr. 2022.
- [6] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, “A Survey of Monte Carlo Tree Search Methods,” *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–43, Mar. 2012.
- [7] R. Coulom, “Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search,” in *Computers and Games*, ser. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2007, pp. 72–83.
- [8] L. Kocsis and C. Szepesvári, “Bandit Based Monte-Carlo Planning,” in *Machine Learning: ECML 2006*, ser. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2006, pp. 282–293.
- [9] D. J. Wu, “Accelerating Self-Play Learning in Go,” in *Proceedings of the AAAI Workshop on Reinforcement Learning in Games*, Nov. 2020.
- [10] G.-C. Pascutto and GitHub contributors, “Leela Zero: A Go program with no human provided knowledge,” Leela Zero, Oct. 2023. [Online]. Available: <https://github.com/leela-zero/leela-zero>
- [11] Y. Tian, J. Ma, Q. Gong, S. Sengupta, Z. Chen, J. Pinkerton, and L. Zitnick, “ELF OpenGo: An analysis and open reimplementation of AlphaZero,” in *Proceedings of the 36th International Conference on Machine Learning*. PMLR, May 2019, pp. 6244–6253.
- [12] T.-R. Wu, T.-H. Wei, and I.-C. Wu, “Accelerating and Improving AlphaZero Using Population Based Training,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 01, pp. 1046–1053, Apr. 2020.
- [13] L. Gary, L. Alexander, H. Folkert, and GitHub contributors, “Leela Chess Zero: A UCI-compliant chess engine designed to play chess via neural network,” LCZero, Oct. 2023. [Online]. Available: <https://github.com/LeelaChessZero/lc0>
- [14] W. Ye, S. Liu, T. Kurutach, P. Abbeel, and Y. Gao, “Mastering Atari Games with Limited Data,” in *Advances in Neural Information Processing Systems*, vol. 34. Curran Associates, Inc., 2021, pp. 25 476–25 488.
- [15] M. Lanctot, E. Lockhart, J.-B. Lespiau, V. Zambaldi, S. Upadhyay, J. Pérolat, S. Srinivasan, F. Timbers, K. Tuyls, S. Omidshafiei, D. Hennes, D. Morrill, P. Muller, T. Ewalds, R. Faulkner, J. Kramár, B. De Vylder, B. Saeta, J. Bradbury, D. Ding, S. Borgeaud, M. Lai, J. Schrittwieser, T. Anthony, E. Hughes, I. Danihelka, and J. Ryan-Davis, “OpenSpiel: A Framework for Reinforcement Learning in Games,” Sep. 2020.
- [16] T. Cazenave, Y.-C. Chen, G.-W. Chen, S.-Y. Chen, X.-D. Chiu, J. Dehos, M. Elsa, Q. Gong, H. Hu, V. Khalidov, C.-L. Li, H.-I. Lin, Y.-J. Lin, X. Martinet, V. Mella, J. Rapin, B. Roziere, G. Synnaeve, F. Teytaud, O. Teytaud, S.-C. Ye, Y.-J. Ye, S.-J. Yen, and S. Zagoruyko, “Polygames: Improved Zero Learning,” Jan. 2020.
- [17] S. Thakoor, S. Nair, and M. Jhunjhunwala, “Learning to play othello without human knowledge,” 2016.
- [18] A. H. Werner Duvaud, “MuZero general: Open reimplementation of MuZero,” 2019. [Online]. Available: <https://github.com/werner-duvaud/muzero-general>
- [19] Y. Niu, Y. Pu, Z. Yang, X. Li, T. Zhou, J. Ren, S. Hu, H. Li, and Y. Liu, “Lightzero: A unified benchmark for monte carlo tree search in general sequential decision scenarios,” in *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023.
- [20] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.
- [21] C. D. Rosin, “Multi-armed bandits with episode context,” *Annals of Mathematics and Artificial Intelligence*, vol. 61, no. 3, pp. 203–230, Mar. 2011.
- [22] A. Fawzi, M. Balog, A. Huang, T. Hubert, B. Romera-Paredes, M. Barekatin, A. Novikov, F. J. R. Ruiz, J. Schrittwieser, G. Swirszcz, D. Silver, D. Hassabis, and P. Kohli, “Discovering faster matrix multiplication algorithms with reinforcement learning,” *Nature*, vol. 610, no. 7930, pp. 47–53, Oct. 2022.
- [23] D. J. Mankowitz, A. Michi, A. Zhernov, M. Gelmi, M. Selvi, C. Paduraru, E. Leurent, S. Iqbal, J.-B. Lespiau, A. Ahern, T. Köppe, K. Millikin, S. Gaffney, S. Elster, J. Broshear, C. Gamble, K. Milan, R. Tung, M. Hwang, T. Cemgil, M. Barekatin, Y. Li, A. Mandhane, T. Hubert, J. Schrittwieser, D. Hassabis, P. Kohli, M. Riedmiller, O. Vinyals, and D. Silver, “Faster sorting algorithms discovered using deep reinforcement learning,” *Nature*, vol. 618, no. 7964, pp. 257–263, Jun. 2023.
- [24] T. Hubert, J. Schrittwieser, I. Antonoglou, M. Barekatin, S. Schmitt, and D. Silver, “Learning and Planning in Complex Action Spaces,” in *Proceedings of the 38th International Conference on Machine Learning*. PMLR, Jul. 2021, pp. 4476–4486.
- [25] I. Antonoglou, J. Schrittwieser, S. Ozair, T. K. Hubert, and D. Silver, “Planning in Stochastic Environments with a Learned Model,” in *International Conference on Learning Representations*, Oct. 2021.
- [26] W. Kool, H. V. Hoof, and M. Welling, “Stochastic Beams and Where To Find Them: The Gumbel-Top-k Trick for Sampling Sequences Without Replacement,” in *Proceedings of the 36th International Conference on Machine Learning*. PMLR, May 2019, pp. 3499–3508, ISSN: 2640-3498.
- [27] Z. Karnin, T. Koren, and O. Somekh, “Almost Optimal Exploration in Multi-Armed Bandits,” in *Proceedings of the 30th International*

Conference on Machine Learning. PMLR, May 2013, pp. 1238–1246, iSSN: 1938-7228.

- [28] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimeshine, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, “PyTorch: An Imperative Style, High-Performance Deep Learning Library,” in *Advances in Neural Information Processing Systems*, vol. 32. Curran Associates, Inc., 2019.
- [29] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, “The arcade learning environment: an evaluation platform for general agents,” *Journal of Artificial Intelligence Research*, vol. 47, no. 1, pp. 253–279, May 2013.
- [30] M. C. Machado, M. G. Bellemare, E. Talvitie, J. Veness, M. Hausknecht, and M. Bowling, “Revisiting the arcade learning environment: evaluation protocols and open problems for general agents,” *Journal of Artificial Intelligence Research*, vol. 61, no. 1, pp. 523–562, Jan. 2018.
- [31] M. Hessel, I. Danihelka, F. Viola, A. Guez, S. Schmitt, L. Sifre, T. Weber, D. Silver, and H. V. Hasselt, “Muesli: Combining Improvements in Policy Optimization,” in *Proceedings of the 38th International Conference on Machine Learning*. PMLR, Jul. 2021, pp. 4214–4226.
- [32] R. Haque, T. H. Wei, and M. Müller, “On the Road to Perfection? Evaluating Leela Chess Zero Against Endgame Tablebases,” in *Advances in Computer Games*, ser. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2022, pp. 142–152.



Ti-Rong Wu (Member, IEEE) received the B.S. degree in computer science from National Chung Cheng University, Chiayi, Taiwan, in 2014, and the Ph.D. degree in computer science from National Chiao Tung University, Hsinchu, Taiwan, in 2020.

He is currently an assistant research fellow with the Institute of Information Science, Academia Sinica, Taipei, Taiwan. His research interests include machine learning, deep reinforcement learning, artificial intelligence, and computer games. He led a team for developing the Go program, CGI, which

won many competitions including second place in the first World AI Open held in Ordos, China, in 2017. He published several papers in top-tier conferences, such as AAAI, ICLR, and NeurIPS.



Hung Guei (Member, IEEE) received the B.S. degree in computer science from National Central University, Taoyuan, Taiwan, in 2015, and the Ph.D. degree in computer science from National Yang Ming Chiao Tung University, Hsinchu, Taiwan, in 2023.

He is currently a postdoctoral scholar with the Institute of Information Science, Academia Sinica, Taipei, Taiwan. His research interests include reinforcement learning, artificial intelligence, and computer games. His research achievements include a

state-of-the-art game-playing program for the game of 2048, which is the best-performing program based on reinforcement learning as of 2023.



Pei-Chiun Peng is currently working toward the M.S. degree in computer science with the Department of Computer Science, National Yang Ming Chiao Tung University, Hsinchu, Taiwan.

His research interests include deep reinforcement learning and computer games.



Po-Wei Huang is currently working toward the M.S. degree in computer science with the Department of Computer Science, National Yang Ming Chiao Tung University, Hsinchu, Taiwan.

His research interests include deep reinforcement learning and computer games.



Ting Han Wei received the Ph.D. degree in computer science from National Chiao Tung University, Hsinchu, Taiwan, in 2019.

He is currently a Professor at the School of Informatics, Kochi University of Technology, Japan. His research interests include computer games, reinforcement learning, heuristic search, and AI safety. His publications include papers in top-tier conferences, such as AAAI, IJCAI, NeurIPS, and ICLR.



Chung-Chin Shih (Member, IEEE) received the B.S. degree in computer science from National Chiao Tung University, Hsinchu, Taiwan, in 2011, and earned his Ph.D. degree in computer science from National Yang Ming Chiao Tung University, Hsinchu, Taiwan, in 2023.

He is currently a postdoctoral scholar with the Institute of Information Science, Academia Sinica, Taipei, Taiwan. His publications include papers in top-tier conferences, such as AAAI, ICLR, and NeurIPS.

His research interests include machine learning, reinforcement learning, and computer games.



Yun-Jui Tsai is currently working toward the M.S. degree in computer science with the Department of Computer Science, National Yang Ming Chiao Tung University, Hsinchu, Taiwan.

His research interests include deep reinforcement learning and computer games.

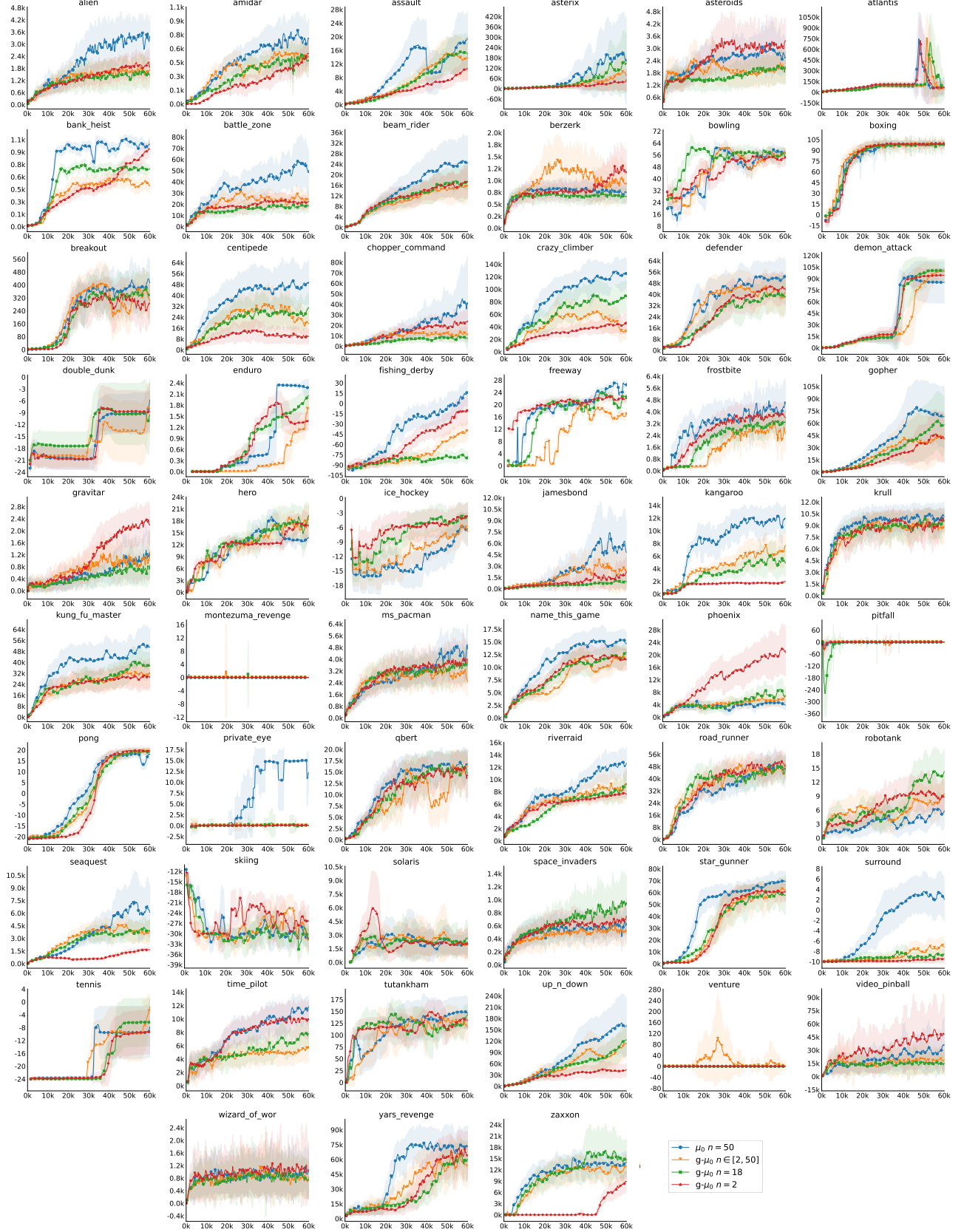


Fig. 12. Training curves on 57 Atari games of different zero-knowledge learning algorithms. The x-axis represents the number of neural network training steps, while the y-axis represents the average returns collected from the last 100 training episodes. The shaded area indicates standard deviation.

TABLE III
SCORES ON 57 ATARI GAMES OF DIFFERENT ZERO-KNOWLEDGE LEARNING ALGORITHMS

Game	μ_0 $n = 50$	$g\text{-}\mu_0$ $n = 18$	$g\text{-}\mu_0$ $n = 2$	$g\text{-}\mu_0$ $n \in [2, 50]$
alien	3,384.00	1,461.30	1,873.60	1,724.70
amidar	710.25	475.70	546.91	517.58
assault	19,447.85	14,933.13	10,786.07	13,870.61
asterix	160,700.00	149,249.50	51,530.00	78,716.00
asteroids	2,549.40	2,100.30	3,382.10	2,034.00
atlantis	70,064.00	64,986.00	62,005.00	49,899.00
bank_heist	998.70	697.40	934.50	494.50
battle_zone	50,130.00	18,530.00	22,250.00	24,570.00
beam_rider	24,393.68	17,331.04	17,370.70	15,816.62
berzerk	741.50	663.80	1,187.80	945.50
bowling	57.30	55.07	54.39	56.75
boxing	99.23	98.14	99.01	98.20
breakout	413.02	333.43	289.65	304.23
centipede	49,823.07	30,749.87	9,946.83	19,242.80
chopper_command	40,159.00	7,578.00	24,153.00	11,245.00
crazy_climber	126,874.00	89,117.00	47,722.00	32,886.00
defender	52,140.00	38,230.50	43,240.50	41,418.50
demon_attack	85,108.10	100,499.85	94,344.80	99,637.15
double_dunk	-5.76	-9.60	-8.60	-7.82
enduro	2,274.20	2,052.87	1,368.13	1,702.76
fishing_derby	17.02	-77.88	-9.55	-41.32
freeway	26.52	22.54	21.97	16.92
frostbite	4,530.00	3,331.30	3,589.00	2,503.70
gopher	67,909.00	57,815.40	40,903.40	40,373.20
gravitar	1,138.50	847.50	2,184.00	1,294.00
hero	13,786.05	19,352.80	16,683.65	18,418.80
ice_hockey	-6.32	-3.66	-3.61	-6.91
jamesbond	4,788.00	984.50	2,181.50	2,347.00
kangaroo	11,984.00	4,876.00	1,978.00	7,671.00
krull	10,134.10	9,567.80	9,606.50	8,811.30
kung_fu_master	51,312.00	37,164.00	29,010.00	31,432.00
montezuma_revenge	0.00	0.00	0.00	0.00
ms_pacman	4,705.00	3,797.70	3,892.30	2,456.50
name_this_game	14,633.20	12,875.80	11,462.40	11,469.80
phoenix	3,837.30	6,964.30	20,722.00	6,631.20
pitfall	0.00	0.00	0.00	-0.56
pong	18.04	19.15	19.80	19.36
private_eye	12,132.96	98.12	63.91	39.68
qbert	16,654.75	14,637.00	14,203.50	16,211.50
riverraid	12,596.00	9,231.60	7,653.40	9,180.80
road_runner	47,020.00	46,111.00	48,345.00	49,357.00
robotank	5.82	14.25	8.81	9.48
seaquest	6,094.40	3,877.80	1,626.60	3,913.00
skiing	-31,808.33	-31,568.41	-26,459.35	-31,117.25
solaris	2,090.80	2,637.20	2,202.60	1,993.40
space_invaders	635.10	931.20	748.50	512.80
star_gunner	69,601.00	57,910.00	60,586.00	61,437.00
surround	2.03	-8.53	-9.45	-7.17
tennis	-7.77	-6.30	-9.27	-1.49
time_pilot	11,836.00	7,786.00	9,908.00	5,675.00
tutankham	146.33	142.76	130.77	117.39
up_n_down	161,069.70	122,743.00	44,026.50	114,536.10
venture	0.00	0.00	0.00	2.00
video_pinball	36,401.50	15,225.94	48,304.05	18,657.30
wizard_of_wor	1,001.00	748.00	1,078.00	829.00
yars_revenge	73,230.71	60,261.63	63,747.76	53,542.43
zaxxon	14,062.00	14,707.00	8,896.00	12,956.00
human normalized mean	485.20%	395.87%	341.29%	359.97%