symmetry

MDPI

Article

Efficient Parallel Design for Self-Play in Two-Player Zero-Sum Games

Hongsong Tang [1,†], Bo Chen [2,†], Yingzhuo Liu [2,†], Kuoye Han [3], Jingqian Liu [4] and Zhaowei Qu [1,*,†]

[1] School of Science, Beijing University of Posts and Telecommunications, Beijing 100876, China; ths@bupt.edu.cn
[2] School of Artificial Intelligence, Beijing University of Posts and Telecommunications, Beijing 100876, China; chenbo@bupt.edu.cn (B.C.); liuyingzhuo86@bupt.edu.cn (Y.L.)
[3] Information Science Academy (ISA), China Electronics Technology Group Corporation (CETC), Beijing 100043, China; kuoyehan@hotmail.com
[4] Chinatelecom Group Corporation, Beijing 100032, China; liujq2@chinatelecom.cn
* Correspondence: quzw@bupt.edu.cn
† Current address: Beijing Key Laboratory of Network System and Network Culture, Beijing University of Posts and Telecommunications, Beijing 100876, China.

Abstract: Self-play methods have achieved remarkable success in two-player zero-sum games, attaining superhuman performance in many complex game domains. Parallelizing learners is a feasible approach to handle complex games. However, parallelizing learners often leads to the suboptimal exploitation of computational resources, resulting in inefficiencies.This paper introduces the Mixed Hierarchical Oracle (MHO), which is designed to enhance training efficiency and performance in complex two-player zero-sum games. MHO efficiently leverages interaction data among parallelized solvers during the Parallelized Oracle (PO) process, while employing Model Soups (MS) to consolidate fragmented computational resources and Hierarchical Exploration (HE) to balance exploration and exploitation. These carefully designed enhancements for parallelized systems significantly improve the training performance of self-play. Additionally, MiniStar is introduced as an open source environment focused on small-scale combat scenarios, developed to facilitate research in self-play algorithms. The MHO is evaluated on both the AlphaStar888 matrix game and MiniStar environment, and ablation studies further demonstrates its effectiveness in improving the agent's decision-making capabilities. This work highlight the potential of the MHO to optimize compute resource utilization and improve performance in self-play methods.

Keywords: game theory; reinforcement learning; deep learning; multi-agent system

check for updates

Academic Editor: Aram Azati Saharian (Saharyan)

Received: 25 December 2024
Revised: 30 January 2025
Accepted: 5 February 2025
Published: 7 February 2025

Citation: Tang, H.; Chen, B.; Liu, Y.; Han, K.; Liu, J.; Qu, Z. Efficient Parallel Design for Self-Play in Two-Player Zero-Sum Games. *Symmetry* **2025**, *17*, 250. https://doi.org/10.3390/sym17020250

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Multi-agent reinforcement learning (MARL) [1] presents unique challenges due to the interdependence of agents' actions, which causes the environment to appear non-stationary from the perspective of each agent [2,3]. Key difficulties in MARL include coordination and equilibrium selection, especially in competitive scenarios. These challenges often hinder convergence, destabilize learning processes, and complicate efficient policy space exploration.

Two-player zero-sum games are a prominent setting within MARL due to their analytical tractability and symmetry, making them a central focus of both theoretical and applied research. In this context, self-play has emerged as a promising approach to address the inherent complexities of MARL in competitive scenarios. By training an agent against copies or past versions of itself [4,5], self-play provides a stable and tractable learning framework.

This approach has found widespread applications across various domains, demonstrating notable success in games such as Go [6–9], chess [8,9], poker [10,11], and video games [12,13], often exceeding human-level performance.

However, traditional self-play methods [14–16] face limitations in complex game environments characterized by vast strategy spaces [17]. The Policy Space Response Oracle (PSRO) [18] extends the Double Oracle (DO) [14] to large-scale games by integrating Reinforcement Learning (RL) [19] to approximate the best responses. The PSRO leverages Empirical Game-Theoretic Analysis (EGTA) [20,21] to study meta-strategies obtained through simulations, and it incorporates a meta-strategy solver to select adversarial strategies. This synthesis of the EGTA with RL enhances strategy selection in self-play while guaranteeing convergence toward the approximate Nash Equilibrium (NE).

Despite these advances, the PSRO continues to face significant computational challenges, with best response oracle calculations in complex scenarios remaining prohibitively expensive. Parallelizing the learning process offers a promising solution to address this challenge [18,22]. Deep Cognitive Hierarchy (DCH) [18] organizes training into a hierarchy, where each learner uses deep RL to train an oracle policy against the NE of a meta-game, improving training efficiency with oracle parallelism. Building upon DCH's foundation, the P-PSRO [23] initializes a population of active policies with assigned hierarchical levels. The P-PSRO improves training efficiency via the parallelized warm-start training of higher-level active policies, employing the NE computed from lower-level policies. A key innovation of the P-PSRO is its optimization of the training hierarchy itself, alleviating the limitation of DCH, which needs to predetermine the number of PSRO iterations. The rectified PSRO (PSRO-rN) [24] is another PSRO of the parallel oracle, where each oracle is trained with the policies it currently beats, reducing redundant policies and improving training efficiency.

Nevertheless, resource utilization and training efficiency remain significant bottlenecks. Current methods for oracle parallelization typically distribute computational resources uniformly across all oracles, leading to suboptimal efficiency. Moreover, in the P-PSRO, the data generated from interactions between high-level active policies and low-level active policies are not utilized for training the low-level active policies, further reducing training efficiency. Additionally, there remain untapped opportunities to optimize parallelized RL oracles and fully realize their potential.

To address these challenges, this paper introduces the MHO framework, which advances parallel oracle methods through three synergistic innovations. First, the MHO employs the PO, which concurrently trains oracles across hierarchical levels, leveraging cross-level interaction data. This approach enhances training efficiency by using high-level vs. low-level policy trajectories to train both participating oracles. Second, the framework employs parameter fusion techniques, MS, to consolidate scattered training resources and initialize new oracles through strategic combinations of existing policy parameters. This approach mitigates cold-start inefficiencies while ensuring competitive viability against lower-level policies. Third, the MHO introduces a hierarchical mechanism, HE, which adjusts exploration–exploitation tradeoffs across hierarchy levels, with higher-level policies emphasizing exploration and lower-level policies focusing on exploitation.

This paper also introduces an open source environment for MiniStar, a variant of SMACv2 [25]. SMACv2 serves as a popular benchmark in MARL [26–28], but it only allows for controlling agents while relying on a built-in bot for the opposing side, thus preventing self-play between two learning agents. To overcome this limitation, MiniStar extends SMACv2 by permitting control over both sides, removing the need for a built-in bot and enabling self-play. By narrowing the scope to tactical engagements rather than full-length strategies, MiniStar decreases the complexity tied to long-horizon decision-making and highlights the core aspects of self-play algorithms without extensive RL optimization.

The principal contributions of this work can be summarized as follows:

- A formal analysis and empirical validation of the inefficiencies in current parallelized best response oracle systems.
- The MHO framework, improving training efficiency and strategic performance in complex games through optimized parallelization.
- MiniStar, a purpose-built benchmark environment for self-play research in tactical combat scenarios.

The remainder of this paper is structured as follows. Section 2 reviews the related work on self-play methods and simulation environments. Section 3 introduces the necessary theoretical preliminaries. Section 4 details the MHO framework, elaborating on its three key components—PO, MS, and HE. Section 5 provides extensive empirical evaluations, including benchmark experiments in AlphaStar888 and MiniStar, with comprehensive ablation studies to assess the impact of each MHO component. Finally, Section 6 concludes the paper with a summary of the findings and a discussion of future research directions.

2. Related Works

2.1. Self-Play Methods

In vanilla self-play [4], agents are trained by repeatedly playing against their latest versions. Fictitious Self-Play (FSP) [11] enables agents to play against their past selves to learn optimal strategies. Neural fictitious self-play [15] is a modern variant that combines FSP with deep learning techniques, using neural networks to approximate the best response. Prioritized fictitious self-play [13] utilizes a preference function to assign higher selection probabilities to higher-priority agents. The DO [14] method approximates the NE in zero-sum games by iteratively creating and solving a series of sub-games with a restricted set of pure strategies. The PSRO [18] is a generalization of the DO, using RL as an oracle to enable decision-making in complex gaming environments. It introduces the concept of a meta-strategy solver to assist in the selection of adversarial strategies, which guarantees convergence to an approximate NE.

To improve the efficiency of self-play training, distributed RL has been integrated into the oracle computation, significantly accelerating the learning process. Distributed RL utilizes parallelized environment sampling to enhance scalability, allowing agents to train in more complex settings. Techniques such as IMPALA [29] and Ape-X [30] have demonstrated efficient learning across various environments through distributed architectures. MALib [31] introduces a highly optimized computational framework that combines distributed RL with self-play, further improving training efficiency.

Unlike distributed RL, which parallelizes environment sampling, DCH [18] parallelizes the oracle computation in self-play and enhances training efficiency through training multiple oracles. The P-PSRO [23] builds upon DCH by refining its parallelization strategy and eliminating the requirement to predetermine the number of PSRO iterations, which is a limitation of DCH. The PSRO-rN [24] introduces an alternative form of parallelized best response oracle training, where each oracle is specifically trained against the policies it currently defeats, promoting greater strategic diversity.

2.2. Self-Play Simulation Environment

Compared to traditional board and card games, simulation environments are typically characterized by real-time operations, long periods, and higher complexity of environmental state transitions, such as StarCraft II [32], Google Research Football (GRF) [33], Dota 2 [12], and Honor of Kings [34–36]. These environments present agents with real-time, partially observable settings requiring continuous decision making over extended time horizons. Agents must handle large, continuous action spaces and deal with uncertainties

introduced by dynamic opponents and environments. The complexity and high dimensionality of these environments necessitate extensive RL and engineering optimization before effective self-play can be conducted. For instance, AlphaStar [13] combines RL, self-play, and imitation learning [37] to achieve master-level performance using vast computational resources. OpenAI Five [12] demonstrates that self-play could be scaled to achieve superhuman performance in Dota 2 by training agents in a massively parallel framework. Similarly, Honor of Kings requires extensive feature processing and complex engineering optimizations to facilitate effective self-play training [34–36]. Even relatively small-scale GRF environments demand substantial pretraining, including imitation learning and curriculum learning, before self-play can be conducted successfully [38,39].

In contrast, lightweight multi-agent environments such as SMAC [40] and SMACv2 [25] provide a more accessible testing ground. However, these environments do not natively support self-play, as they are designed for training agents against built-in bots, rather than learning through direct competition between two learning agents. This limitation restricts their applicability in self-play research.

Motivated by this gap, MiniStar is introduced as a lightweight simulation environment designed to facilitate self-play training. MiniStar is a variant of SMACv2 that retains the core characteristics of SMACv2 while overcoming its limitation of training only against built-in bots. By enabling direct competition between learning agents, MiniStar provides a practical and efficient platform for studying self-play without the need for extensive RL engineering.

3. Preliminaries

3.1. Two-Player Normal-Form Games

A two-player normal-form game [41] is characterized by the tuple $(\mathcal{A}, \mathbf{U})$, where $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ represents the action sets for each player $i \in \{1, 2\}$, and $\mathbf{U} = (u_1, u_2)$ denotes their respective utility functions. Formally, for each player i, the utility function $u_i : \mathcal{A}_1 \times \mathcal{A}_2 \to \mathbb{R}$ assigns a real-valued payoff to every possible action pair.

Players aim to maximize their expected utility by choosing a mixed strategy $\pi_i \in \Delta(\mathcal{A}_i)$, where $\Delta(\mathcal{A}_i)$ denotes the set of probability distributions over $\mathcal{A}_i$. For notational convenience, the opponent of player i is denoted as $-i$. The best response to an opponent's mixed strategy π_{-i} is the strategy $\mathcal{BR}(\pi_{-i})$ that maximizes player i's utility as follows:

$$\mathcal{BR}(\pi_{-i}) = \arg\max_{\pi_i} G_i(\pi_i, \pi_{-i}), \tag{1}$$

where $G_i(\pi_i, \pi_{-i})$ denotes the expected utility for player i when using policy π_i against the opponent's policy π_{-i}.

3.2. Meta-Strategy

The concept of a meta-game extends the game to a higher level of abstraction by considering a population of policies $\Pi_i = \pi_i^1, \pi_i^2, \ldots$ for each player i. In this context, choosing an action corresponds to selecting a specific policy from the set Π_i. The interactions within this expanded policy space are captured by the payoff matrix $M_{\Pi_i, \Pi_{-i}}$, where $M_{\Pi_i, \Pi_{-i}}[j, k] = G_i(\pi_i^j, \pi_{-i}^k)$.

In the meta-game, a meta-strategy σ_i represents a mixed strategy over the policy set Π_i, assigning probabilities to each policy in the set. Meta-games are often open-ended because an infinite number of mixed strategies can be constructed from the available policies.

In self-play methods, each player i maintains a set of strategies Π_i for themselves and observes the opponent's strategy set Π_{-i}. This framework allows for the construction of a meta-strategy σ_i to elucidate the dynamics between players' policies. The meta-strategy σ_i

for player i is derived from various solvers such as NE, fictitious play [42], or prioritized fictitious self-play.

When a new policy π_i' is introduced, the framework recalculates the best response, often referred to as the oracle. If the oracle is determined through RL, the best response is represented as follows:

$$\mathcal{BR}(\sigma_{-i}) = \max_{\pi_i'} \sum_j \sigma_{-i}^j E_{\pi_i', \pi_{-i}^j}[R], \tag{2}$$

where R denotes the RL reward, typically configured in a zero-sum setting, and $E_{\pi_i', \pi_{-i}^j}[R]$ represents the expected reward when player i uses policy π_i' against the opponent's policy π_{-i}^j.

To quantify how far the joint meta-strategy profile $\sigma = (\sigma_1, \sigma_2)$ is from a NE, exploitability is measured using NashConv as follows [18]:

$$\text{Expl}(\sigma) = \sum_{i=1}^{2} [G_i(\mathcal{BR}(\sigma_{-i}), \sigma_{-i}) - G_i(\sigma_i, \sigma_{-i})], \tag{3}$$

where $\mathcal{BR}(\sigma_{-i})$ denotes the best response oracle to the opponent's meta-strategy σ_{-i}. When the exploitability reaches zero, the joint meta-strategy profile σ corresponds to a NE.

4. Methodology

This section introduces the MHO framework, which significantly enhance the training efficiency and performance of RL agents in self-play algorithms, particularly in complex environments. The MHO comprises the following three key components: PO, MS, and HE. These components aim to address the training efficiency of the policy learning process from different perspectives—sample utilization, cold-start issues, and exploration mechanisms—and are ultimately integrated to work synergistically. The detailed framework is illustrated in Figure 1, and the step-by-step procedure is provided in Algorithm 1.

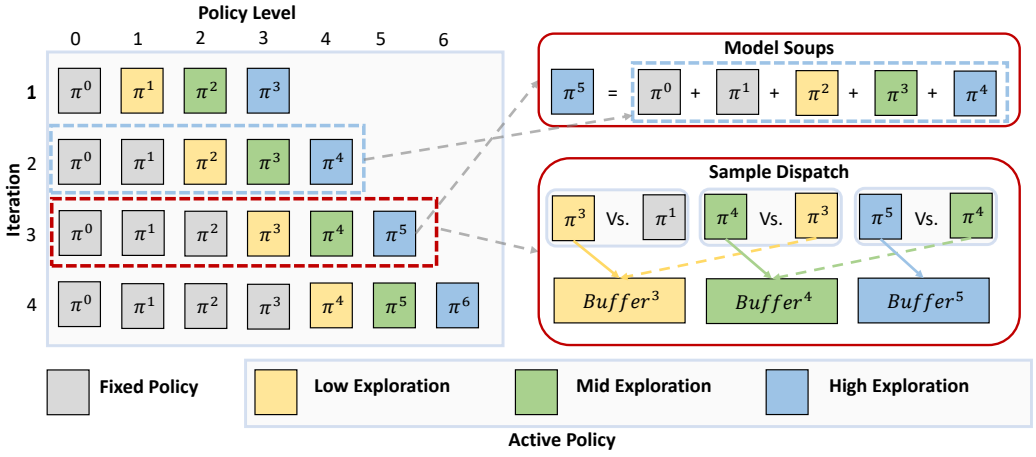


Figure 1. The overall framework diagram of MHO. The policy in MHO consists of fixed policy which is the fixed and active policy is being trained. Active policy is a set of parallel hierarchical policies. Higher-level policies are more exploratory in training and lower-level policies are more exploitative. After the lowest-level policy (yellow in the figure) completes training, it becomes a fixed policy. A new active policy is added as the highest-level policy (blue in the figure) and initialized by the lower-level policy using the MS method. After the higher-level policy finishes fighting against the lower-level active policy, the samples are learned by the respective level active policy, instead of discarding the samples of the lower-level active policy [23].

Algorithm 1 Mixed Hierarchical Oracle (MHO)

Initialize the policy set Π^f.
/ Parallelized Oracle. */*
Parallelize n active policies to initialize the active policy set Π^a.
Compute the meta-strategy σ_i^j using fictitious self-play for the j-th active policy π_i^j.
for epoch in $\{1, 2, ...\}$ **do**
  for player $i \in \{1, 2\}$ **do**
    for many episodes **do**
      / Distributed, simultaneous. */*
      for active policy $\pi_i^j \in \Pi^a$ **do**
        Sample $\pi_{-i} \sim \sigma_{-i}^j$.
        / Hierarchical Exploration. */*
        Train π_i^j using the objective in Equation (7).
      end for
    end for
    if π_i^j converges (plateaus) and is the lowest-level active policy **then**
      Update the fixed policy set: $\Pi_i^f \leftarrow \Pi_i^f \cup \{\pi_i^j\}$.
      Remove the policy from the active set: $\Pi_i^a \leftarrow \Pi_i^a \setminus \{\pi_i^j\}$.
      / Model Soups. */*
      Initialize new active policy π_i^n at a higher level than all existing active policies.
      Update π_i^n via MS (Equation (6)).
      Add the new policy to the active set: $\Pi_i^a \leftarrow \Pi_i^a \cup \{\pi_i^n\}$.
      Update the meta-strategy σ_i for each active policy.
    end if
  end for
end for
Output the current lowest active policy as the final trained policy.

4.1. Parallelized Oracle

In P-PSRO training, two critical issues arise. First, when a high-level active policy samples experience against a lower-level active policy, only the high-level policy's corresponding samples are retained for training; samples that belongs to the lower-level policy are discarded. For instance, as illustrated in Figure 1 (sample dispatch stage), when π^4 plays against π^3, the data generated for π^3 are not allocated to π^3 for learning. Second, each active policy during training is allocated an equal portion of the total computational resources, which reduces training efficiency.

To address the first issue identified in the P-PSRO, this study introduces a PO approach. The PO maintains a fixed policy set (Π^f) and an active policy set (Π^a). Π^f retains the fixed model parameter, while Π^a is trained by n parallel RL workers, each operating at the corresponding hierarchical level among n levels. Each active policy π_i^j is trained against the meta-strategy derived from Π^f and Π^a, which are obtained from lower levels of the hierarchy. When a high-level active policy interacts with a lower-level active policy, all resulting data samples are redistributed to the corresponding active policies. Once the lowest-level active policy π_i^j converges, π_i^j then transitions from Π^a to Π^f, and π_i^{j+1} becomes the new lowest-level active policy. A new active policy π_i^n is initialized at a higher level than all existing active policies, marking the beginning of a new training cycle.

To more clearly illustrate the advantages of the PO over existing parallelized best response oracle systems, particularly the P-PSRO, a formal comparative analysis is conducted. In the PO approach, $w_{\text{active},j}$ denotes the total sampling probability of the j-th policy interacting with all other active policies except itself. Let $m (m < j)$ be the number of active policies and n be the total number of sample environments. For the k-th active

policy, the cumulative increase in the number of samples from other active policies ΔS_k is given by the following:

$$\Delta S_k = \frac{n}{m} \times w_{\text{active},j-m+k}, \tag{4}$$

the total increase number of samples ΔS_{total} is then adjusted as follows:

$$\Delta S_{\text{total}} = \frac{n}{m} \times \sum_{k=1}^{m} w_{\text{active},j-m+k}. \tag{5}$$

During parallel training, each active policy further utilizes the data generated by the high-level policy while training with the fixed policy. Equation (5) demonstrates that the PO approach can leverage a larger volume of data, ΔS_{total}, than existing parallelized methods, without discarding any of them, thereby enhancing sample efficiency during training.

Although the PO approach alters how training data are gathered and redistributed, it does not significantly erode the approximate best response properties. In particular, when FSP is used as the meta-strategy solver, the method approximately retains the same convergence characteristics as standard FSP. This occurs because, as training progresses, the fixed policies dominate the overall strategy distribution.

4.2. Model Soups

To address the second issue in P-PSRO, where allocating equal computational resources leads to proportionally reduced training capacity for each oracle, the model fusion approach, MS, is incorporated to mitigate this limitation. This method effectively consolidates computational resources. Additionally, MS resolve the cold-start problem. When models are initialized from scratch in each new training round, agents may struggle to develop effective policies as opponents become increasingly strong. By employing MS, each oracle begins with a well-established policy foundation, thereby enhancing learning efficiency and overall performance.

Specifically, after each round of training, a new top-level active policy is obtained by the parameter fusion of the lower-level active policies and the fixed policies. This fusion shares the knowledge learned among different active policies, enhancing data utilization and accelerating learning in subsequent training rounds.

In the context of MS, the meta-strategy is employed as a weighted combination of the model parameters from the lower-level policies. Mathematically, for the set of policies $\Pi_i = \{\pi_i^1, \pi_i^2, \ldots, \pi_i^j\}$, and $\theta_{\pi_i^j}$ denotes the parameters of the j-th policy; under the meta-strategy σ_i, the new policy $\theta_{\pi_i^{j+1}}$ is computed as follows:

$$\theta_{\pi_i^{j+1}} = \sum_{k=1}^{j} \sigma_i^{j+1,k} \cdot \theta_{\pi_i^k}. \tag{6}$$

MS mitigate computational fragmentation caused by the PO through the fusion of model parameters across different learners, effectively recombining the split computational resources. This fusion enhances data utilization by sharing knowledge within the policy pool, overcoming the inefficiency of independent learning in parallelized settings.

4.3. Hierarchical Exploration

In self-play algorithms, it is often necessary to truncate the approximate best response operator at each iteration, which can lead to suboptimal training outcomes. To migrate this issue, the exploration mechanism HE is introduced within the PO approach, assigning different exploration factors to various levels of the active policy pool.

Specifically, the highest-level active policies are more inclined to explore during training after initialization, and they gradually shift towards exploitation as training progresses. To achieve this, an entropy regularization term is incorporated into the computation of the best respone oracle. The entropy term encourages exploration by penalizing deterministic behavior in the policy, thus promoting stochasticity during training.

Let $H(\pi)$ represent the entropy of a policy π. The objective function for Equation (2) is modified as follows:

$$\mathcal{BR}(\sigma_{-i}) = \max_{\pi_i'} \sum_j \sigma_{-i}^j E_{\pi_i', \pi_{-i}^j}[R] - \lambda_k H(\pi_i'), \tag{7}$$

here, λ_k is a hyperparameter that controls the strength of the entropy regularization. As training progresses, λ_k decreases in tandem with the shift of policies from high-active to low-active levels, with the highest-level policy having the largest λ_k value and the lowest-level policy having the smallest. This synchronized reduction of λ_k ensures that exploration is encouraged early in the training, while policies progressively focus more on exploitation as they transition towards lower activity levels. This mechanism effectively maintains a balance between exploration and exploitation across different policy tiers, ensuring that agents explore sufficiently in the early stages, while refining and exploiting learned strategies in the later stages.

5. Experiments

5.1. Experimental Setup

To evaluate the effectiveness of the MHO in complex two-player zero-sum games, this study compares the MHO with representative self-play algorithms, including the PSRO [18], P-PSRO [23], PSRO-rN [24], and Self-Play (SP) [16]. The experiments span both matrix games and the MiniStar environment. The hyperparameter settings for the experiments are provided in Appendix A.

All experiments were implemented in Python (version 3.8.0), utilizing PyTorch (version 1.10.0) for neural network training and optimization, NumPy and Pandas for data processing, and Matplotlib for result visualization. The MiniStar environment was managed using Python to facilitate efficient simulation and interaction. Experiments were conducted on a server equipped with an Intel(R) Xeon(R) CPU E5-2690 v4 processor, 220 GB of RAM, and a NVIDIA RTX 3090 GPU with 24 GB of dedicated memory. This computational setup ensured efficient model training, simulation, and evaluation, accommodating for the intensive requirements of RL and extensive self-play iterations.

5.2. Experimental Environment

5.2.1. Martix Game

AlphaStar888 [43] is an empirical game derived from the solution process of StarCraft II [13], featuring a payoff table involving 888 RL policies. It can be viewed as a zero-sum symmetric two-player game with only one state. In this state, there are 888 legal actions, and any mixed strategy corresponds to a discrete probability distribution over these actions.

5.2.2. Simulation Scenario

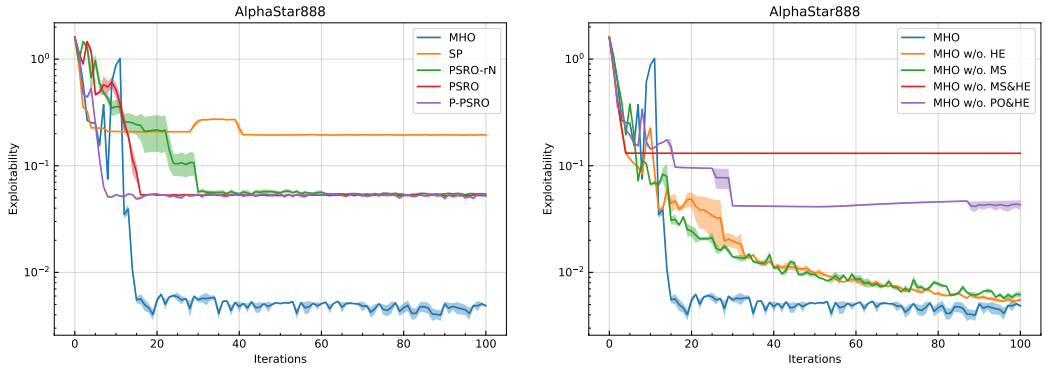
The MiniStar environment is a simplified version of StarCraft II [32], designed specifically for skirmish scenarios and self-play research. By focusing on localized battle control rather than the full spectrum of StarCraft II gameplay, which includes resource management, mission planning, and large-scale battle control, MiniStar allows agents to concentrate on the microlevel manipulation of decision-making actions. This targeted approach reduces

the complexity of the environment, enabling the more efficient development of zero-sum game algorithms in focused combat situations. In SMAC [40] and SMACv2 [25], agents control one faction while the opposing faction is managed by a built-in bot, and there is no support for agents to control both factions. By contrast, MiniStar extends SMACv2 by allowing agents to control both factions simultaneously in a self-play setting, eliminating the need for built-in bots.The MiniStar environment is an open source environment and available at https://github.com/QrowBranwen/MiniStar (accessed on 7 January 2025).

In the experiment, each of the three races is tested under the 5v5 matchmaking mode. For Zerg, they are zergling, hydralisk, and baneling; for Terran, they are marine, marauder, and medivac; and for Protoss, they are stalkers, zealots, and colossi. The three racial unit weights relative to a fixed unit order are [0.45, 0.45, 0.1], and birth locations are randomized for the Surround and Reflect scheme. Every algorithm employs the Multi-Agent Proximal Policy Optimization (MAPPO) algorithm [44] as the oracle, relying exclusively on self-play for 20 million steps and without using data from matches against the built-in AI.

5.3. Results and Analysis

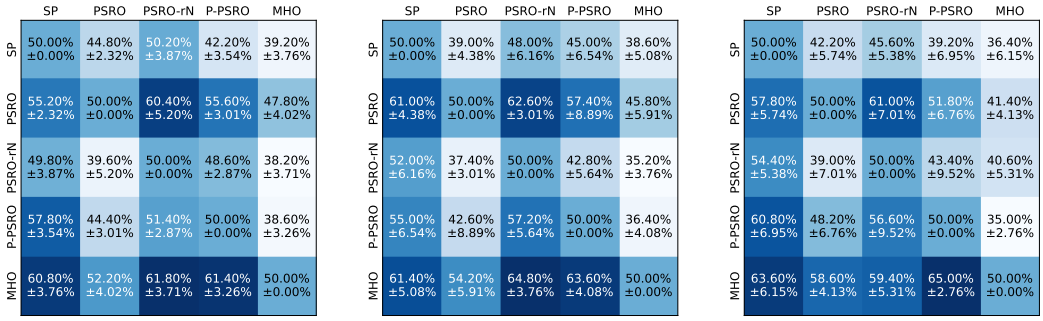
For AlphaStar888, the experimental results are shown in Figure 2. The experiments recorded the exploitability [18] of each algorithm's current policy set, which was then plotted against iterations, with iterations on the horizontal axis and exploitability on the vertical axis. The results demonstrate that the MHO algorithm achieves the best performance.



(**a**) (**b**)

Figure 2. The experiments in AlphaStar888. (**a**) Main experimental results comparing different algorithms, with exploitability plotted against training iterations. (**b**) Ablation experiment on AlphaStar888 comparing the performance of the MHO, with exploitability plotted against training iterations.

In MiniStar, to benchmark performance, pairwise matches are conducted among the models produced by each algorithm, and the resulting win rate matrix is reported. This direct comparison allows us to quantitatively assess the relative strengths of the different methods. The experimental results are presented in Figure 3. The MHO consistently outperforms the baseline methods (PSRO, PSRO-rN, P-PSRO, and SP) in all three races (Protoss, Zerg, and Terran) in 5v5 settings, achieving the highest expected win rate. In addition, throughout the training process, each algorithm's policy is periodically evaluated against built-in AI. The win rate curves from these evaluations serve as an indirect measure of the training progress, providing further insights into how each algorithm evolves over time. The experimental results are presented in Figure 4. Compared to other parallelized methods (P-PSRO, PSRO-rN), the MHO exhibits faster convergence and superior final performance, demonstrating improved training efficiency and strategic effectiveness. Overall, the MHO maintains a higher win rate.



(**a**) Protoss 5v5 (**b**) Zerg 5v5 (**c**) Terran 5v5

Figure 3. Win rate matrices comparing MHO, PSRO, P-PSRO, PSRO-rN, and self-play in the MiniStar environment for three races: (**a**) Protoss 5v5, (**b**) Zerg 5v5, and (**c**) Terran 5v5. Each cell shows the row player's expected payoff against the column player's strategy. Larger positive values (darker coloration) indicate stronger performance of the row strategy against the column strategy.

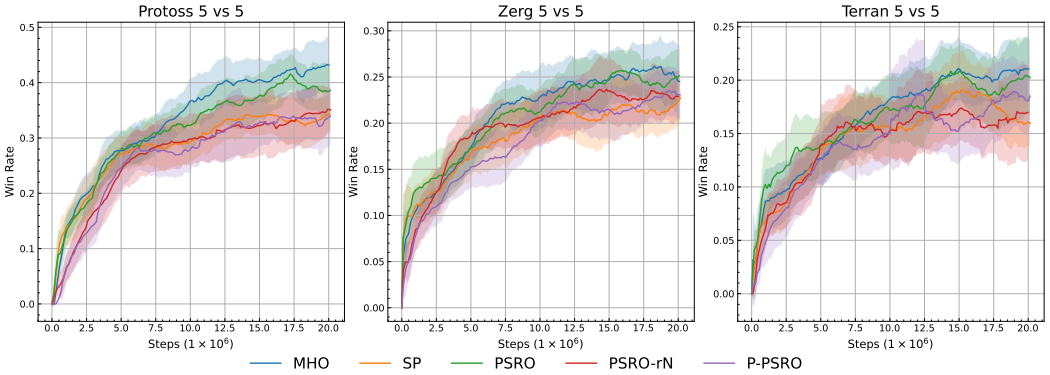


Figure 4. Winning curves of different algorithms against built-in AI during training. The horizontal axis represents the number of training steps, and the vertical axis denotes the average win rate

5.4. Ablation Studies

The contribution of each component of the MHO, including the PO, MS, and HE, is rigorously evaluated through a series of ablation experiments conducted on both the AlphaStar888 benchmark and the MiniStar environment. This study follows a subtractive methodology as follows: starting from the full MHO configuration (PO + MS + HE), we then remove individual components or their combinations. This strategy helps us precisely isolate the impact of each module and ensure that performance differences can be clearly attributed to the presence or absence of specific elements.

The study considers the following variants:

- MHO (Full):PO + MS + HE.
- MHO w/o. HE: Remove HE.
- MHO w/o. MS: Remove MS.
- MHO w/o. MS&HE: Remove both MS and HE.
- MHO w/o. PO&HE: Remove both PO and HE.

These subsets allow us to examine the effect of eliminating key components individually and in combination, thereby testing each module's unique and synergistic contributions.

5.4.1. Parallelized Oracle

Considering AlphaStar888 is a single-step matrix game, the discussion of sample utilization is focused on the MiniStar environment. In Figure 5, MHO w/o. MS&HE exhibits slower convergence in the early stages. A similar phenomenon appears when comparing the PSRO and the P-PSRO in Figure 4, where the P-PSRO exhibits slower

convergence as well. These findings confirm that dividing total computational resources among multiple oracles can reduce early-phase training efficiency.

To further examine this effect more closely, the PO is introduced into the PSRO (denoted as PO-PSRO) for a more granular comparison, as shown in Figure 6. The results indicate that the PO-PSRO attains faster convergence in the early training stages. This improvement arises because the PO approach leverages more samples, enhancing the efficiency of the training process.

From the perspective of the final results, the ablation experiments in AlphaStar888 and MiniStar, Figures 2b and 7, show that the MHO w/o. HE outperforms the MHO w/o. PO&HE, and the MHO w/o. HE achieves better results than the MHO w/o. PO&HE.

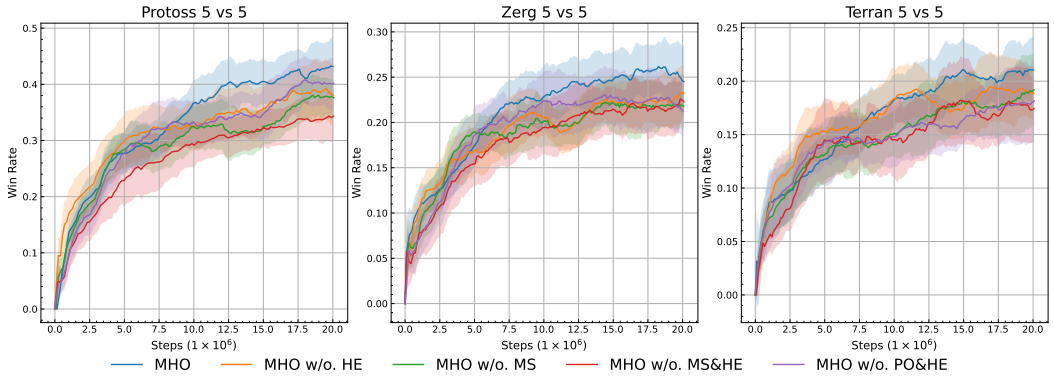


Figure 5. Ablation experiments against built-in AI in the MiniStar environment. Winning curves of various ablated versions of the MHO against the built-in AI. The x-axis indicates the number of training steps, while the y-axis denotes the corresponding win rate.

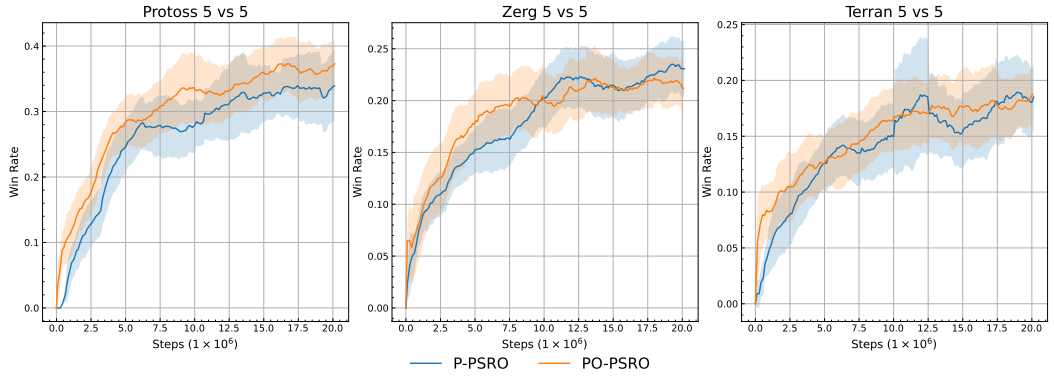
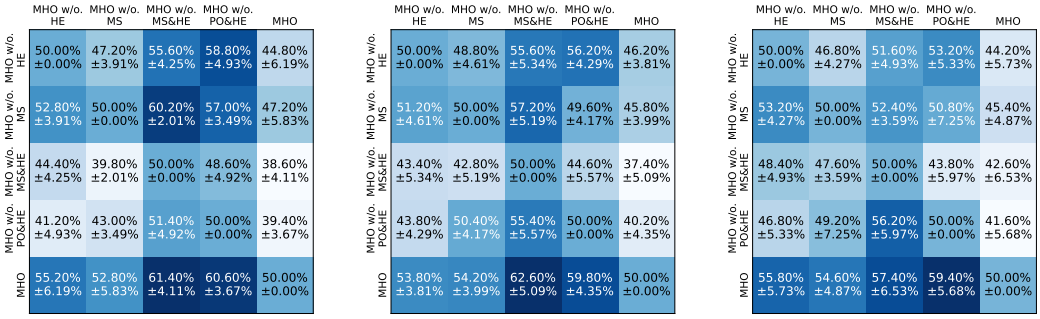


Figure 6. Average win rate against built-in AI: P-PSRO vs. PO-PSRO. The horizontal axis denotes the number of training steps, and the vertical axis represents the average win rate against the built-in AI.

5.4.2. Model Soups

In Figure 5, when the PO is paired with the MS component (MHO w/o. HE), it achieves faster early-stage convergence than the MHO w/o. PO&HE. This observation indicates that MS indirectly consolidate computational resources otherwise fragmented by PO, thus retaining parallelization benefits while mitigating the PO's inherent drawbacks.

As shown in Figures 2b, 5, and 7, comparisons between the MHO and the MHO w/o.MS, as well as between the MHO w/o. HE and the MHO w/o. MS&HE, consistently demonstrate the effectiveness of the MS component. By integrating knowledge acquired through parallelized learning, MS indirectly boost data utilization and enhance overall training efficiency and performance.



(**a**) Protoss 5v5 (**b**) Zerg 5v5 (**c**) Terran 5v5

Figure 7. Ablation experiments in the MiniStar environment. Pairwise win rate matrices evaluating MHO without certain components in 5v5 combat scenarios of MiniStar. Subfigures show (**a**) Protoss 5v5, (**b**) Zerg 5v5, and (**c**) Terran 5v5. Each cell shows the row player's expected payoff against the column player's strategy. Larger positive values (darker coloration) indicate stronger performance of the row strategy against the column strategy.

5.4.3. Hierarchical Exploration

As shown in Figures 2b, 5, and 7, comparisons between the MHO and the MHO w/o. HE, as well as between the MHO w/o. MS and the MHO w/o. MS&HE, demonstrate that incorporating HE achieves superior performance. HE promotes broader strategy exploration during the early stages of training and, in the later phases, shifts toward exploitation to improve final decision-making performance.

5.4.4. Ablation Studies Summary

When all three components are present, their benefits combine synergistically. The MHO achieves superior results, faster improvements, lower exploitability, and stronger final performance than any configuration missing one or more modules.

6. Conclusions

This paper introduces the MHO to address suboptimal resource utilization in parallelized RL oracles. The MHO integrates three key techniques, the PO, MS, and HE, to significantly improve training efficiency and model performance. Specifically, the PO increases the amount of interaction data available, thus enhancing learning speed. MS effectively amalgamate knowledge from multiple hierarchical policies, mitigating the inefficiencies caused by uniform resource allocation while simultaneously alleviating the cold-start problem. HE further refines the training process by promoting broader strategy exploration during the initial phases, followed by more fine-grained exploitative learning in later stages, culminating in a better performance.In the AlphaStar888 matrix game and MiniStar environment, the MHO demonstrates superior performance over multiple baseline self-play algorithms. Ablation studies confirm the mutually complementary nature of the three core components, showing that they collectively drive efficient learning in complex adversarial scenarios. Overall, the MHO provides a scalable, high-efficiency solution for two-player zero-sum games in large-scale, high-dimensional settings, offering effective perspectives for future research in RL and self-play research.

Furthermore, to address the shortage of suitable simulation scenarios for game research in this domain, this paper presents the MiniStar environment. By focusing on simplified battle engagements rather than the full complexity of real-time strategy games, MiniStar substantially reduces the engineering overhead typically required in zero-sum research, thus serving as a lightweight and flexible platform for the broader community.

Future work will focus on the following areas: The efficient integration of parallel training in self-play and distributed RL parallel sampling, rather than treating them as

independent components, to further enhance training efficiency and enable the application of the framework to larger-scale training scenarios. Further optimization of the parallelized framework while ensuring strict theoretical guarantees for convergence. Enhancement of the MiniStar environment, as the current scenarios do not fully emphasize environmental factors such as terrain, which may limit the exploration of strategy diversity. Future improvements will introduce more diverse training scenarios, providing a richer simulation environment for two-player zero-sum game research.

Author Contributions: Conceptualization, H.T. and B.C.; methodology, H.T., B.C., and Y.L.; software, H.T. and B.C.; validation, B.C. and Y.L.; formal analysis, K.H.; investigation, J.L.; resources, Z.Q.; data curation, H.T.; writing—original draft preparation, H.T. and B.C.; writing—review and editing, H.T., K.H., and Z.Q.; visualization, H.T. and B.C.; supervision, Z.Q. and J.L.; project administration, K.H. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding

Data Availability Statement: The data used in this paper are all simulated. The original simulated environment presented in the study is included in the article, further inquiries can be directed to the first author.

Conflicts of Interest: Author Kuoye Han was employed by the company Information Science Academy (ISA), China Electronics Technology Group Corporation (CETC). Author Jingqian Liu was employed by the company Chinatelecom Group Corporation. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Appendix A

Table A1. Hyper-parameter settings for AlphaStar888.

Setting	Value	Description
Oracle Function	Best Response	Function for obtaining oracles.
Learning Rate	0.5	Learning rate for agents.
Improvement Threshold	0.03	Convergence criterion.
PSRO Meta-Strategy	Nash Equilibrium	Solves the NE-Strategy.
MHO Meta-Strategy	Fictitious Self-play	Solves the NE-Strategy.
Threads in Pipeline	3	Number of parallel learners.
Iterations	100	Training iterations.
Random Seeds	5	Number of random seeds.
λ_k	[0.1, 0.15, 0.2]	Weighting factor for entropy.

Table A2. Hyper-parameter settings for Ministar.

Setting	Value	Description
Oracle	MAPPO	The reinforcement learning algorithm used for Oracle.
Training Steps	20M	Total number of environment steps for training.
Self-Play Mode	5v5	Each match features two teams of 5 units each.
Unit Composition	[0.45, 0.45, 0.1]	Ratio of 3 unit types (e.g., Zealot/Stalker/Colossus).
Spawn Scheme	Surround & Reflect	Randomized initial positions for both factions.
PSRO Meta-Strategy	Nash Equilibrium	Meta-Strategy Solver.
MHO Meta-Strategy	Fictitious Self-play	Meta-Strategy Solver.
Discount Factor (γ)	0.99	Discounting for future rewards.
Learning Rate	5×10^{-4}	Adam optimizer step size.
PPO Clip Parameter	0.2	Clipping range for ratio.
Entropy Coefficient	0.01	Encourages exploration in MAPPO.

Table A2. *Cont.*

Setting	Value	Description
Threads in Pipeline	2	Number of parallel learners
λ_k	[0.008, 0.012]	Weighting factor for entropy in MAPPO.
Number of Actors	16	Number of enviroment in distributed RL.
Batch Size	2048	Number of sampled transitions per update.
Number of Mini-Batches	1	Number of mini-batches per epoch in MAPPO.
PPO Epochs	5	Number of times each sample is reused.
GAE Lambda	0.95	Exponential decay factor for GAE advantage.
Value Loss Weighting	1.0	Trade-off coefficient for value function loss.
Random Seed	5	Number of random seeds used.

References

1. Albrecht, S.V.; Christianos, F.; Schäfer, L. *Multi-Agent Reinforcement Learning: Foundations and Modern Approaches*; MIT Press: Cambridge, MA, USA, 2024.
2. Mahajan, A.; Rashid, T.; Samvelyan, M.; Whiteson, S. Maven: Multi-agent variational exploration. In Proceedings of the 33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, BC, Canada, 8–14 December 2019; Volume 32.
3. Rashid, T.; Samvelyan, M.; De Witt, C.S.; Farquhar, G.; Foerster, J.; Whiteson, S. Monotonic value function factorisation for deep multi-agent reinforcement learning. *J. Mach. Learn. Res.* **2020**, *21*, 1–51.
4. Some studies in machine learning using the game of checkers. *IBM J. Res. Dev.* **1959**, *3*, 210–229. [CrossRef]
5. Bansal, T.; Pachocki, J.; Sidor, S.; Sutskever, I.; Mordatch, I. Emergent complexity via multi-agent competition. *arXiv* **2017**, arXiv:1710.03748.
6. 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* **2016**, *529*, 484–489. [CrossRef]
7. Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; et al. Mastering the game of go without human knowledge. *Nature* **2017**, *550*, 354–359. [CrossRef]
8. 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* **2017**, arXiv:1712.01815.
9. Schrittwieser, J.; Antonoglou, I.; Hubert, T.; Simonyan, K.; Sifre, L.; Schmitt, S.; Guez, A.; Lockhart, E.; Hassabis, D.; Graepel, T.; et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature* **2020**, *588*, 604–609. [CrossRef] [PubMed]
10. Moravčík, M.; Schmid, M.; Burch, N.; Lisỳ, V.; Morrill, D.; Bard, N.; Davis, T.; Waugh, K.; Johanson, M.; Bowling, M. Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. *Science* **2017**, *356*, 508–513. [CrossRef]
11. Heinrich, J.; Lanctot, M.; Silver, D. Fictitious self-play in extensive-form games. In Proceedings of the International Conference on Machine Learning, Lille, France, 6–11 July 2015; pp. 805–813.
12. Berner, C.; Brockman, G.; Chan, B.; Cheung, V.; Debiak, P.; Dennison, C.; Farhi, D.; Fischer, Q.; Hashme, S.; Hesse, C.; et al. Dota 2 with large scale deep reinforcement learning. *arXiv* **2019**, arXiv:1912.06680.
13. Vinyals, O.; Babuschkin, I.; Czarnecki, W.M.; Mathieu, M.; Dudzik, A.; Chung, J.; Choi, D.H.; Powell, R.; Ewalds, T.; Georgiev, P.; et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature* **2019**, *575*, 350–354. [PubMed]
14. McMahan, H.B.; Gordon, G.J.; Blum, A. Planning in the presence of cost functions controlled by an adversary. In Proceedings of the 20th International Conference on Machine Learning (ICML-03), Washington, DC, USA, 21–24 August 2003; pp. 536–543.
15. Heinrich, J.; Silver, D. Deep reinforcement learning from self-play in imperfect-information games. *arXiv* **2016**, arXiv:1603.01121.
16. Hernandez, D.; Denamganaï, K.; Gao, Y.; York, P.; Devlin, S.; Samothrakis, S.; Walker, J.A. A generalized framework for self-play training. In Proceedings of the 2019 IEEE Conference on Games (CoG), London, UK, 20–23 August 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 1–8.
17. Yang, Y.; Luo, J.; Wen, Y.; Slumbers, O.; Graves, D.; Ammar, H.B.; Wang, J.; Taylor, M.E. Diverse auto-curriculum is critical for successful real-world multiagent learning systems. *arXiv* **2021**, arXiv:2102.07659.
18. Lanctot, M.; Zambaldi, V.; Gruslys, A.; Lazaridou, A.; Tuyls, K.; Pérolat, J.; Silver, D.; Graepel, T. A unified game-theoretic approach to multiagent reinforcement learning. In Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA, 4–9 December 2017; Volume 30.
19. Sutton, R.S. Reinforcement learning: An introduction. In *A Bradford Book*; MIT Press: Cambridge, MA, USA, 2018.
20. Wellman, M.P. Methods for empirical game-theoretic analysis. In Proceedings of the AAAI, Boston, MA, USA, 16–20 July 2006; Volume 980, pp. 1552–1556.
21. Wellman, M.P.; Tuyls, K.; Greenwald, A. Empirical game-theoretic analysis: A survey. *arXiv* **2024**, arXiv:2403.04018.

22. Bighashdel, A.; Wang, Y.; McAleer, S.; Savani, R.; Oliehoek, F.A. Policy Space Response Oracles: A Survey. *arXiv* **2024**, arXiv:2403.02227.
23. McAleer, S.; Lanier, J.B.; Fox, R.; Baldi, P. Pipeline psro: A scalable approach for finding approximate nash equilibria in large games. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 20238–20248.
24. Balduzzi, D.; Garnelo, M.; Bachrach, Y.; Czarnecki, W.; Perolat, J.; Jaderberg, M.; Graepel, T. Open-ended learning in symmetric zero-sum games. In Proceedings of the International Conference on Machine Learning. Long Beach, CA, USA, 9–15 June 2019; pp. 434–443.
25. Ellis, B.; Cook, J.; Moalla, S.; Samvelyan, M.; Sun, M.; Mahajan, A.; Foerster, J.; Whiteson, S. Smacv2: An improved benchmark for cooperative multi-agent reinforcement learning. In Proceedings of the 37th Conference on Neural Information Processing Systems (NeurIPS 2023) Track on Datasets and Benchmarks, New Orleans, LA, USA, 10–16 December 2023; Volume 36.
26. Beck, J.; Vuorio, R.; Liu, E.Z.; Xiong, Z.; Zintgraf, L.; Finn, C.; Whiteson, S. A survey of meta-reinforcement learning. *arXiv* **2023**, arXiv:2301.08028.
27. Rutherford, A.; Ellis, B.; Gallici, M.; Cook, J.; Lupu, A.; Ingvarsson, G.; Willi, T.; Khan, A.; de Witt, C.S.; Souly, A.; et al. Jaxmarl: Multi-agent rl environments in jax. *arXiv* **2023**, arXiv:2311.10090.
28. Zhong, Y.; Kuba, J.G.; Feng, X.; Hu, S.; Ji, J.; Yang, Y. Heterogeneous-agent reinforcement learning. *J. Mach. Learn. Res.* **2024**, *25*, 1–67.
29. Espeholt, L.; Soyer, H.; Munos, R.; Simonyan, K.; Mnih, V.; Ward, T.; Doron, Y.; Firoiu, V.; Harley, T.; Dunning, I.; et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In Proceedings of the International Conference on Machine Learning, Stockholm, Sweden, 10–15 July 2018; pp. 1407–1416.
30. Horgan, D.; Quan, J.; Budden, D.; Barth-Maron, G.; Hessel, M.; Van Hasselt, H.; Silver, D. Distributed prioritized experience replay. *arXiv* **2018**, arXiv:1803.00933.
31. Zhou, M.; Wan, Z.; Wang, H.; Wen, M.; Wu, R.; Wen, Y.; Yang, Y.; Yu, Y.; Wang, J.; Zhang, W. MALib: A parallel framework for population-based multi-agent reinforcement learning. *J. Mach. Learn. Res.* **2023**, *24*, 1–12.
32. Vinyals, O.; Ewalds, T.; Bartunov, S.; Georgiev, P.; Vezhnevets, A.S.; Yeo, M.; Makhzani, A.; Küttler, H.; Agapiou, J.; Schrittwieser, J.; et al. Starcraft ii: A new challenge for reinforcement learning. *arXiv* **2017**, arXiv:1708.04782.
33. Kurach, K.; Raichuk, A.; Stańczyk, P.; Zajac, M.; Bachem, O.; Espeholt, L.; Riquelme, C.; Vincent, D.; Michalski, M.; Bousquet, O.; et al. Google research football: A novel reinforcement learning environment. In Proceedings of the AAAI Conference on Artificial Intelligence, New York, NY, USA, 7–12 February 2020; Volume 34, pp. 4501–4510.
34. Ye, D.; Chen, G.; Zhang, W.; Chen, S.; Yuan, B.; Liu, B.; Chen, J.; Liu, Z.; Qiu, F.; Yu, H.; et al. Towards playing full moba games with deep reinforcement learning. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 621–632.
35. Ye, D.; Chen, G.; Zhao, P.; Qiu, F.; Yuan, B.; Zhang, W.; Chen, S.; Sun, M.; Li, X.; Li, S.; et al. Supervised learning achieves human-level performance in moba games: A case study of honor of kings. *IEEE Trans. Neural Netw. Learn. Syst.* **2020**, *33*, 908–918. [CrossRef]
36. Wei, H.; Chen, J.; Ji, X.; Qin, H.; Deng, M.; Li, S.; Wang, L.; Zhang, W.; Yu, Y.; Linc, L.; et al. Honor of kings arena: An environment for generalization in competitive reinforcement learning. *Adv. Neural Inf. Process. Syst.* **2022**, *35*, 11881–11892.
37. Hussein, A.; Gaber, M.M.; Elyan, E.; Jayne, C. Imitation learning: A survey of learning methods. *ACM Comput. Surv. (CSUR)* **2017**, *50*, 21. [CrossRef]
38. Lin, F.; Huang, S.; Pearce, T.; Chen, W.; Tu, W.W. Tizero: Mastering multi-agent football with curriculum learning and self-play. *arXiv* **2023**, arXiv:2302.07515.
39. Huang, S.; Chen, W.; Zhang, L.; Li, Z.; Zhu, F.; Ye, D.; Chen, T.; Zhu, J. TiKick: Towards Playing Multi-agent Football Full Games from Single-agent Demonstrations. *arXiv* **2021**, arXiv:2110.04507.
40. Samvelyan, M.; Rashid, T.; De Witt, C.S.; Farquhar, G.; Nardelli, N.; Rudner, T.G.; Hung, C.M.; Torr, P.H.; Foerster, J.; Whiteson, S. The starcraft multi-agent challenge. *arXiv* **2019**, arXiv:1902.04043.
41. Fudenberg, D.; Tirole, J. *Game Theory*; MIT Press: Cambridge, MA, USA, 1991.
42. Brown, G.W. Iterative solution of games by fictitious play. *Act. Anal. Prod Alloc.* **1951**, *13*, 374.
43. Czarnecki, W.M.; Gidel, G.; Tracey, B.; Tuyls, K.; Omidshafiei, S.; Balduzzi, D.; Jaderberg, M. Real world games look like spinning tops. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 17443–17454.
44. Yu, C.; Velu, A.; Vinitsky, E.; Wang, Y.; Bayen, A.; Wu, Y. The surprising effectiveness of ppo in cooperative, multi-agent games. *arXiv* **2021**, arXiv:2103.01955.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.