Mastering Multi-Drone Volleyball through Hierarchical Co-Self-Play Reinforcement Learning

Ruize Zhang, Sirui Xiang, Zelai Xu, Feng Gao, Shilong Ji, Wenhao Tang, Wenbo Ding, Chao Yu†, Yu Wang†
Tsinghua University
jimmyzhangruize@gmail.com, {yuchao,yu-wang}@mail.tsinghua.edu.cn

Abstract: In this paper, we tackle the problem of learning to play 3v3 multi-drone volleyball, a new embodied competitive task that requires both high-level strategic coordination and low-level agile control. The task is turn-based, multi-agent, and physically grounded, posing significant challenges due to its long-horizon dependencies, tight inter-agent coupling, and the underactuated dynamics of quadrotors. To address this, we propose Hierarchical Co-Self-Play (HCSP), a hierarchical reinforcement learning framework that separates centralized high-level strategic decision-making from decentralized low-level motion control. We design a three-stage population-based training pipeline to enable both strategy and skill to emerge from scratch without expert demonstrations: (I) training diverse low-level skills, (II) learning high-level strategy via self-play with fixed low-level skills, and (III) joint fine-tuning through co-self-play. Experiments show that HCSP achieves superior performance, outperforming non-hierarchical self-play and rule-based hierarchical baselines with an average 82.9% win rate and a 71.5% win rate against the two-stage variant. Moreover, co-self-play leads to emergent team behaviors such as role switching and coordinated formations, demonstrating the effectiveness of our hierarchical design and training scheme. The project page is at https://sites.google.com/view/hi-co-self-play.

Keywords: Hierarchical Reinforcement Learning, Self-Play, Multi-Agent System

1 Introduction

Competitive tasks have long served as benchmarks for progress in artificial intelligence. Landmark results have been achieved in domains such as Go [1], poker [2], and real-time strategy games [3], where agents learn to plan, adapt, and compete under structured rules. As research moves from virtual environments to the physical world, robot sports–structured, rule-based competitions involving physical agents–have emerged as a promising frontier for embodied intelligence. Examples include robot soccer [4, 5], table tennis [6, 7], and multi-drone pursuit-evasion [8], which combine high-level strategy with low-level motion control in physically grounded settings.

In this paper, we tackle a new embodied competitive task proposed by the VolleyBots testbed [9]: 3v3 multi-drone volleyball. This task exemplifies the structure of a robot sport–well-defined objectives, explicit rules, and head-to-head competition–while presenting a set of unique and underexplored challenges. Each team must coordinate three quadrotors to rally a ball over a net, switching roles dynamically between offense and defense in a turn-based fashion. The environment is highly dynamic and demands precise timing, agile 3D maneuvering, and strategic team-level behavior. The turn-based nature of ball exchange introduces long-horizon temporal dependencies; the multi-agent setting requires tightly coupled tactics; and the underactuated dynamics of quadrotors call for fine-grained, reactive motor skills. Compounding these challenges, no expert demonstrations are

† Corresponding authors.

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

available, ruling out imitation learning [10] and requiring agents to discover both motion primitives and team strategies from scratch.

To address these challenges, we propose Hierarchical Co-Self-Play (HCSP), a hierarchical reinforcement learning (HRL) framework that separates strategic reasoning from motor control. The policy is decomposed into a high-level strategy responsible for team tactics and multiple low-level skills responsible for drone-specific motion control. The high-level strategy is a centralized multi-layer perceptron (MLP) with three output heads, each producing commands for one drone, enabling synchronized decision-making across the team. The low-level policies are independent across drones and operate at a higher frequency to ensure responsive, fine-grained control. To reflect the turn-based structure of the game, the high-level strategy is only activated upon discrete game events–when the ball is struck or crosses the net–while low-level control runs continuously between these moments.

Training such a system end-to-end is challenging due to the tight coupling between high-level strategy and low-level control. To address this, we propose a three-stage training pipeline that enables both skills and strategies to emerge from scratch, without relying on any expert supervision. In Stage I, we train diverse low-level motion skills using reward shaping and policy chaining, an iterative training method that ensures smooth transitions between temporally adjacent skills. In Stage II, we freeze the low-level controllers and train the high-level strategy via population-based self-play and sample reallocation, allowing the team to learn fundamental cooperative and adversarial behaviors. In Stage III, we jointly fine-tune both high- and low-level policies through co-self-play, enabling mutual adaptation and further performance gains.

Our experiments show that the policy trained with HCSP significantly outperforms both standard self-play and rule-based hierarchical baselines. Beyond overall performance, we conduct a detailed analysis of the three-stage training pipeline. In particular, the final co-self-play stage (Stage III) leads to the emergence of sophisticated team behaviors, such as specialized formations and adaptive role switching between passing and attacking, which are absent in earlier stages. This results in a 71.5% win rate over Stage II, validating the benefits of end-to-end hierarchical adaptation. Ablation studies further confirm the contribution of HCSP's core components, including policy chaining in Stage I and sample reallocation in Stage II, each playing a critical role in enabling the emergence of effective and coordinated multi-agent behavior.

Our main contributions are as follows:

- We conduct the first systematic study of the 3v3 multi-drone volleyball task, which combines strategic team play, precise motion control, and high-speed aerial agility in a turn-based, multi-agent physical environment.
- We propose Hierarchical Co-Self-Play (HCSP), a hierarchical reinforcement learning method that integrates a centralized high-level strategy for team tactics and decentralized low-level skills for agile control.
- We introduce a three-stage population-based training pipeline that enables the emergence of both motion skills and tactical strategies without expert demonstrations. The final co-self-play stage fosters emergent behaviors such as role switching and formation specialization.
- In extensive experiments, HCSP outperforms both non-hierarchical self-play and rule-based hierarchical baselines, achieving an 82.9% average win rate against the baselines and a 71.5% win rate against the two-stage variant.

2 Preliminaries

2.1 3v3 Multi-Drone Volleyball Task

VolleyBots [9] introduces a drone volleyball testbed built on the OmniDrones [11] simulator, leveraging NVIDIA Isaac Sim's [12, 13] high-fidelity physics engine for efficient training. Our work focuses on VolleyBots's 3v3 competitive task, which slightly simplifies real-world volleyball rules

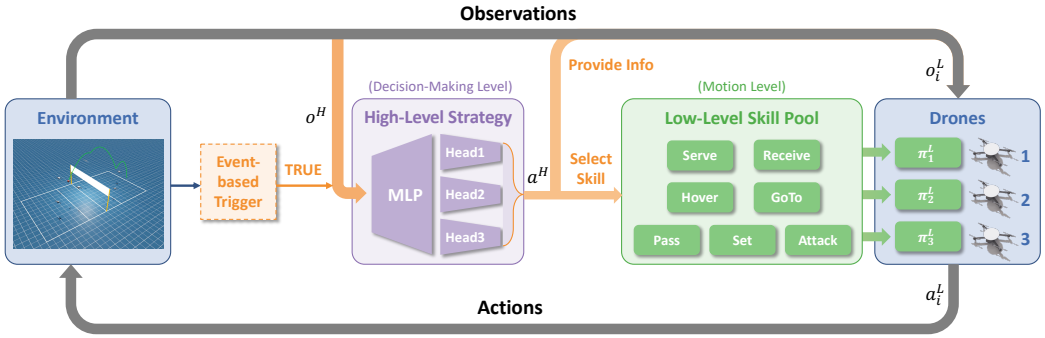


Figure 1: HCSP architecture: an event-driven high-level strategy handles strategic decisions, while multiple low-level skills manage continuous control. Each drone i runs its low-level skill at 50Hz, taking local observation o_i^L and high-level tactical parameters to produce continuous action a_i^L. The high-level strategy activates only on discrete events (racket strike or ball crossing the net), observes o^H, and outputs a^H to choose each drone's skill and supply tactical information. It is implemented as a shared MLP with three output heads, one per drone.

for drone implementation. Two teams of three drones each compete on a scaled-down 6m × 12m court while maintaining the official net height of 2.43m. Each team must return the ball across the net within three consecutive hits using coordinated actions. A team loses points for failing to return the ball, hitting out-of-bounds, violating turn-based interaction rules, or having drones collide with the ground. Each episode lasts a maximum of 15 seconds, starting with a randomized serving team. The episode ends when either one team scores or the time limit is reached. The simulation employs Iris quadrotors [14] equipped with horizontal 0.2m-radius rackets with 0.8 restitution coefficient. The drones are controlled via Per-Rotor Thrust (PRT) inputs at 50 Hz and receive state-based observations. The volleyball itself is modeled as a 5g sphere with 0.1m radius and 0.8 restitution coefficient to simulate realistic collision dynamics. See Appendix A for more details.

2.2 Markov Game and MARL

We model the 3v3 multi-drone volleyball task as a Partially Observable Markov Game (POMG) [15, 16], defined by the tuple $\langle \mathcal{N}, \mathcal{S}, \{\mathcal{A}_i\}_{i\in\mathcal{N}}, \{\mathcal{O}_i\}_{i\in\mathcal{N}}, P, \{R_i\}_{i\in\mathcal{N}}, \gamma \rangle$. $\mathcal{N} = \{1, \cdots, n\}$ denotes the set of agents. $\mathcal{S}$ represents the global state space. Each agent i operates within an action space $\mathcal{A}_i$ and receives observations from an observation space $\mathcal{O}_i$. The state transition probability function $P : \mathcal{S} \times \{\mathcal{A}_i\}_{i\in\mathcal{N}} \times \mathcal{S} \to [0, 1]$ defines the probability of transitioning from one state to another given the joint actions of all agents. $R_i : \mathcal{S} \times \mathcal{A}_i \to \mathbb{R}$ is the reward function of agent i, and $\gamma \in [0, 1]$ is the discount factor determining the importance of future rewards. The 3v3 multi-drone volleyball task is a symmetric game. Both teams share the same continuous observation and action space and payoff structures (win or lose). Multi-agent reinforcement learning (MARL) provides a natural framework for addressing Markov Games. In MARL, at each time step t, every agent i receives an observation $o_{i,t}$ from the environment and selects an action $a_{i,t}$ according to its policy $\pi_i \in \Pi_i : \mathcal{O}_i \times \mathcal{A}_i \to [0, 1]$. Given the joint action of all agents, the environment transitions from the current state s_t to the subsequent state s_{t+1} based on the transition function P and returns an immediate reward $r_{i,t+1}$ to each agent i. Ultimately, the goal of agent i is to learn a policy π_i that maximize its expected discounted cumulative reward $\mathbb{E}_{\pi_i}[\sum_{t=0}^{\infty} \gamma^t r_{i,t+1}]$.

3 Method

The 3v3 multi-drone volleyball task presents dual challenges of precise motion control and coordinated team strategy. To address these requirements, we propose *Hierarchical Co-Self-Play (HCSP)* (architecture in Fig. 1), a hierarchical reinforcement learning (HRL) method that decomposes the overall policy into an event-driven high-level strategy for team tactics and low-level skills for individual maneuvers. HCSP's training pipeline comprises three stages: (I) We train each low-level skill

Table 1: Description of seven low-level skills acquired in stage I and their corresponding tactical observations provided by the high-level strategy.

Skill Name	Skill Description	Observation Provided by the High-Level Strategy
Serve	Serve the ball toward the opponent's court	Target ball position (x, y, z)
Receive	Receive the serve ball and pass it to the setter	/
Pass	Pass the ball to the setter	Pass from court side $\{\text{left, right}\}$
Set	Place the ball optimally for the attacker	/
Attack	Strike the ball toward the opponent's court	Attack direction $\{\text{left, right}\}$
Hover	Maintain at a fixed position and altitude	Preceding skill $\{\textit{Serve}, \textit{Receive}, \textit{Pass}, \textit{Set}, \textit{Attack}\}$
GoTo	Move the drone to a specified location	Target drone position (x, y, z)

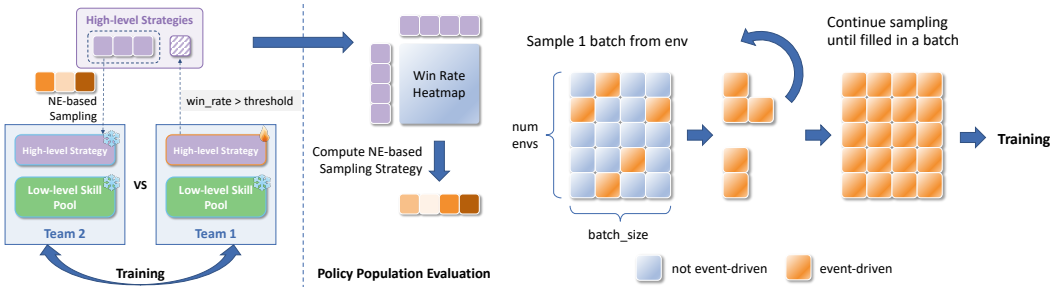
via RL using policy chaining, which addresses the challenge of executing temporally adjacent primitives smoothly by incrementally training each new skill in the context of previously learned ones, resulting in a robust pool of motion primitives. (II) With the low-level skill pool frozen, we pretrain the high-level strategy through population-based team-level self-play, enabling it to acquire initial cooperative and competitive tactics. (III) We utilize co-self-play to co-optimize both skill levels through team-level self-play, where the high-level strategy dynamically selects from continuously improving low-level skills while simultaneously refining their execution.

3.1 Stage I: Low-Level Skill Acquisition

Low-level skills focus on controlling individual drones to execute specific motion primitives without tactical reasoning. Each drone is regarded as a low-level agent. Agent i's low-level skill observation $o_i^L (i \in \{1, \cdots, 6\})$ consists solely of ball dynamics and the drone's own state, together with high-level tactical parameters, but excludes information about other drones. Following VolleyBots [9], which shows that Per-Rotor Thrust (PRT) slightly outperforms Collective Thrust and Body Rates (CTBR) in multi-drone volleyball by leveraging independent rotor control for greater agility, we adopt PRT as our low-level action space $a_i^L (i \in \{1, \cdots, 6\})$. Low-level actions are executed at 50 Hz. Additionally, we apply reward shaping to supply dense feedback and accelerate skill acquisition. For more details on the low-level skill design, see Appendix B.

Due to the challenge of ensuring smooth transitions between temporally adjacent low-level primitives, especially when control switches across drones or skill types, we employ policy chaining to iteratively train each low-level skill for seamless execution under the event-driven high-level strategy. For example, under policy chaining, the *Attack* skill is trained following a teammate's execution of the *Set* skill, enabling coordinated multi-agent behaviors. This approach enhances intra-drone motion continuity and inter-drone cooperation. Our experiments show that policy chaining significantly improves the reliability of sequential skill execution and accelerates learning. Detailed analysis is provided in Sec 4.3.1. All low-level skills are optimized using Proximal Policy Optimization (PPO) [17], resulting in a diverse skill pool summarized in Table 1.

3.2 Stage II: High-Level Strategy Pretraining



(a) Full training schedule for Stage II. (b) Sample reallocation for high-level strategy training.

Figure 2: Illustrations of the high-level strategy pretraining stage (Stage II).

High-level strategies enable team-level decision-making, with each team modeled as a centralized high-level agent. The observation o_j^H ($j \in \{1, 2\}$) includes the states of all six drones (three teammates and three opponents), the ball dynamics, and other game-relevant features. At each decision point, the high-level strategy outputs an action a_j^H that assigns a low-level skill and corresponding tactical parameters to each of its three drones. To support coordinated control, we implement the policy network as a shared multilayer perceptron (MLP) with three output heads, one per drone. Because the task is turn-based, high-level strategies are executed only at discrete events, such as when a racket hits the ball or when the ball crosses the net, indicating a possible change in team roles. The reward for the high-level strategy π_j^H at timestep t is defined as:

$$r_{j,t}^H = c_1 \times \text{win\_or\_lose}_j + c_2 \times \text{racket\_hit\_ball}_j, \tag{1}$$

where $\text{win\_or\_lose}_j = 1$ if team j wins, -1 if it loses, and 0 otherwise; $\text{racket\_hit\_ball}_j = 1$ if any racket of team j contacts the ball at timestep t, and 0 otherwise.

In this stage, we freeze the low-level skills and pretrain the high-level strategy through team-level iterative self-play [18] (Fig. 2a). As the game is symmetric, both teams share the same high-level strategy population, and we train only one side against the other. At each iteration, a candidate high-level strategy competes against frozen opponents sampled from the population according to the Nash equilibrium (NE)-based distribution derived from the current win-rate matrix. When the candidate's win rate exceeds a threshold or a maximum step count is reached, it is added to the shared population. We then perform pairwise evaluations of all population policies to produce an updated win-rate matrix, recompute the NE-based sampling distribution, and repeat the cycle. Since the high-level strategy is event-driven, its action steps are temporally sparse, leading to high-variance gradients and unstable learning. To mitigate this issue, we apply *sample reallocation* (Fig. 2b), which extracts transitions at event-triggered timesteps and attributes cumulative rewards over the intervals between events. These event-triggered transitions are stored in a separate buffer for training. The effectiveness of sample reallocation is analyzed in detail in Sec. 4.3.2.

3.3 Stage III: Co-Self-Play

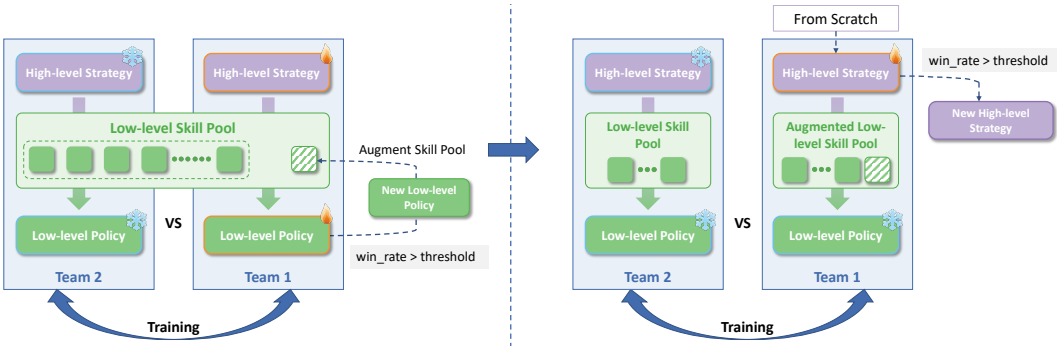
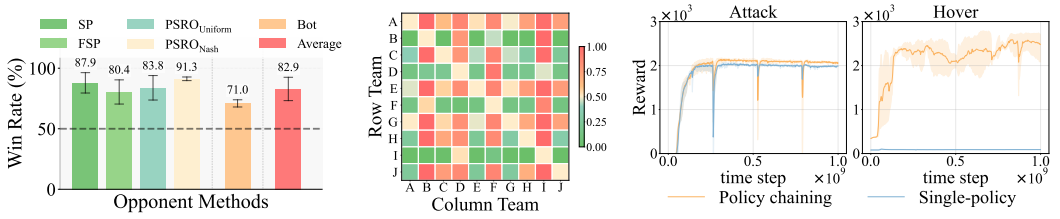


Figure 3: Illustration of the co-self-play stage (Stage III).

As high-level strategies can motivate low-level skill development and, in turn, improvements in low-level skills inform high-level decision-making, co-self-play is essential to iteratively optimize both layers. In Stage III, we jointly refine both high- and low-level policies (Fig. 3). Similar to Stage II, we train only one team at a time against a frozen opponent, with the candidate high- and low-level skills learning together while the opponent's policies remain frozen. Whenever the candidate's win rate exceeds a threshold, the newly trained low-level policies are added to the low-level pool. As the low-level skill pool has grown, the high-level actor's output layer needs to accommodate the expanded action set. We then freeze the augmented low-level pool and train a new high-level strategy from scratch against the former high-level strategy and the original low-level skill pool.

In Stage III, a key challenge in co-self-play lies in defining appropriate reward signals for low-level skills. A naive choice is to reuse the hand-crafted rewards from Stage I; however, this approach fails to account for two critical issues. First, the task setting fundamentally changes in a game



(a) (b) (c)

Figure 4: Experiment results. (a) HCSP performance against baseline policies. (b) Cross-play win rate heatmap among ten randomly picked policies. (c) Ablation study on policy chaining in Stage I: policy chaining enables a successful transition from *Attack* to *Hover*, whereas single-policy fails.

context—for example, a ball that appears to be landing might still be intercepted by an opponent drone, making previous reward definitions inconsistent or even misleading. Second, Stage I rewards are task-specific and offer limited room for further evolution, thus constraining the emergence of advanced behaviors. Alternatively, manually redesigning reward functions for every skill under game dynamics would demand substantial expert effort and risks introducing bias. To overcome these limitations, we directly adopt the high-level reward as the learning signal for low-level skills during co-training. This approach is entirely free of manual design and promotes the emergence of skill adaptations driven by strategic outcomes, enabling the co-evolution of high-level tactics and low-level execution. Because low-level actions execute at 50 Hz while high-level rewards occur only at discrete events, this reward signal is too sparse for effective low-level learning. To mitigate this, we add a KL-divergence penalty between the current low-level policy π_i^L and its reference policy $\pi_{i,ref}^L$. Due to space limitation, we analyze the impact of this KL penalty in detail in Appendix D.2. The resulting co-self-play reward for low-level agent i on team $j \in \{1, 2\}$ at time t is:

$$r_{i,t}^L = r_{j,t}^H - c_3 \times KL(\pi_i^L || \pi_{i,ref}^L). \tag{2}$$

4 Experiments

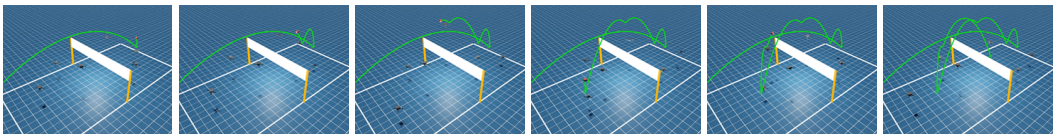
We conduct a series of experiments to evaluate HCSP and elucidate its design. In particular, we seek to answer the following two questions: (1) How does HCSP's performance compare with baseline methods? (2) What is the impact of Stage III co-self-play on overall policy effectiveness? (3) Which HCSP components are the key elements to its success?

Baselines. We evaluate HCSP against five baselines following VolleyBots [9]. Four of them are non-hierarchical game-theoretic self-play methods: Self-Play (SP) [19], Fictitious Self-Play (FSP) [20], and two variants of Policy-Space Response Oracles (PSRO) [18] using either a uniform meta-solver (PSRO$_{\text{Uniform}}$) or a Nash meta-solver (PSRO$_{\text{Nash}}$). The fifth is a rule-based hierarchical policy included in VolleyBots, which we denote as "Bot". See Appendix E for a detailed baseline description.

Evaluation metrics. Head-to-head win rate, evaluated over 500 trajectories per matchup, serves as our primary metric for comparing relative policy strength. However, it may fail to reflect absolute strength due to potential cyclic dominance in non-transitive games. As noted by Liu et al. [21], although Elo rating [22] is commonly used for this purpose, it is sensitive to the policy population. To mitigate this, they propose the Nash-averaging evaluator, which computes a Nash equilibrium over randomly sampled ten policies. We adopt it as our secondary metric in Sec. 4.2.1. All evaluations are averaged over three seeds. Details of implementation can be found in Appendix F.

4.1 Game Results Against Baselines

Fig. 4a shows the win rates when HCSP plays against each of the five baseline methods. HCSP achieves an average win rate of 82.9% across all matchups, demonstrating its strong competitive advantage. Notably, it significantly outperforms all four non-hierarchical game-theoretic methods. Moreover, HCSP attains a 71% win rate against the strongest baseline, the rule-based hierarchical Bot policy provided in VolleyBots [9]. These results indicate that HCSP both outperforms traditional self-play strategies and competes strongly against the handcrafted hierarchical policy.



(a) (b) (c) (d) (e) (f)

Figure 5: Sequence of six temporally sampled frames illustrating an emergent team behavior. (a) The opponent team *passes* the ball to the setter. (b) The opponent team *sets* the ball to the attacker. (c) The opponent team *attacks* the ball towards our side. (d) Our team *passes* the ball to the setter. (e) Our team performs a "dump" shot, sending the ball directly onto the opponent's court. (f) The opponent team fails to return the ball, so our team scores the point.

4.2 Analysis of Stage III Co-Self-Play

4.2.1 Quantitative Analysis

To evaluate the effectiveness of Stage III training, we first conduct a head-to-head tournament between the Stage II and Stage III policies, as shown in the first two columns of Tab. 2. The Stage III policy achieves a 71.5% win rate over the Stage II policy, demonstrating clear improvement under the co-self-play framework. However, this comparison only reflects relative performance between two policies. To assess absolute robustness, we construct a Nash-averaging evaluator following Liu et al. [21]. Specifically, we randomly sample ten policies from all trained populations, including baselines and all HCSP stages, and compute their pairwise win rates (Fig. 4b). We then apply Nash averaging to obtain a mixed evaluation policy, which assigns non-zero weights to three policies {A (0.37), E (0.07), G (0.56)} while assigning zero weight to all others. Finally, we evaluate both the Stage II and Stage III policies against this Nash-averaged opponent. As shown in Tab. 2, the Stage III policy improves its win rate from 31.4% (Stage II) to 47.6%, a 16.2% absolute gain. Notably, the Stage III policy reaches approximately 50% win rate against the Nash-averaged evaluator, indicating competitive performance near the Nash Equilibrium of the sampled strategy set.

Table 2: Win rates of Stage II policy and Stage III policy against different opponents.

	Stage II policy	Stage III policy	Nash-averaged Policy
Stage II policy	50.0 ± 0.0	28.5 ± 8.0	31.4 ± 9.3
Stage III policy	$\mathbf{71.5 \pm 8.0}$	$\mathbf{50.0 \pm 0.0}$	$\mathbf{47.6 \pm 1.0}$

4.2.2 Emergent Low-Level Skill

From a qualitative perspective, the experimental results further show that Stage III's co-self-play induces emergent low-level behaviors that were not designed in the initial stage. In Stage I, policy chaining yields three motion primitives, *Pass*, *Set*, and *Attack*, with high accuracy, so each team always uses three consecutive contacts (*Pass* to the setter drone, *Set* to the attacker drone, *Attack* over the net). During Stage II, with low-level skills frozen, the high-level strategy continues to learns this three-touch sequence. In contrast, after the Stage III, we observe a novel two-touch maneuver, akin to the real-world volleyball "dump" shot, in which the setter itself redirects the ball over the net instead of raising it for a teammate. This emergent "dump" skill requires no additional reward design and illustrates two key effects of co-self-play: low-level primitives can evolve beyond their initial definitions, and the high-level strategy adapts its skill allocation (for example, omitting *Attack* and assigning *Hover* to the would-be attacker drone) to leverage these new behaviors. Fig. 5 presents a case study in which one team uses the emergent "dump" skill to score a point.

4.3 Ablation Study

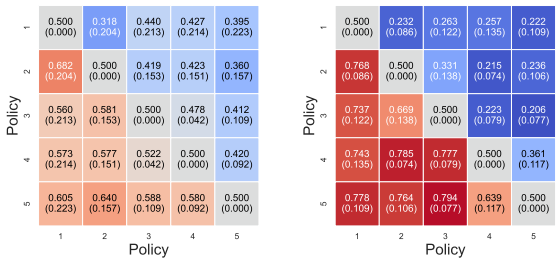
4.3.1 Policy Chaining in Stage I

In Stage I, we employ policy chaining, an iterative training method designed to ensure smooth transitions between temporally adjacent low-level skills. We compare policy chaining with a single-policy approach by training the "*Attacking* the ball to the opponent's side, followed by *Hovering* to maintain stability" behavior under the same reward settings. Through policy chaining, we first train

an *Attack* policy using *Attack*-specific rewards, then train a subsequent *Hover* policy with *Hover*-specific rewards. In contrast, the single-policy approach trains one unified policy using the combined *Attack* and *Hover* rewards. In Fig. 4c, they achieve similar performance in *Attack*. However, only policy chaining successfully learns *Hover* after *Attack*, demonstrating that policy chaining reduces exploration difficulty by decomposing the complex behavior into sequential, specialized policies.

4.3.2 High-level Strategy Sample Reallocation in Stage II

Because the high-level strategy is activated only at discrete time steps, standard minibatch sampling yields sparse effective samples and high variance. To address this, we introduce the sample reallocation procedure that reconstructs each training batch exclusively from event-driven steps (Fig. 2b). We compare standard minibatch sampling to our sample reallocation method by visualizing the high-level strategy's evolution over five self-play iterations using win-rate heatmaps (Figs. 6a and 6b). In these heatmaps, off-diagonal entries reflect each new policy's win rate against its predecessors. Using sample reallocation, these off-diagonal cells become strongly favorable much earlier, indicating faster skill progression, and exhibit lower win-rate variance, indicating more stable training. Also, we compare two policy populations in 500 head-to-head matches averaged over three random seeds, where each side samples a strategy from its population according to its NE–based distribution at the start of each match. The sample reallocation population wins $90.5\% \pm 4.5\%$ of matches against the standard-sampling population.



(a) w/o sample reallocation. (b) w/ sample reallocation.

Figure 6: Comparison of policy population evolution win-rate heatmaps in Stage II.

5 Related Work

The domain of robot sports has become a critical frontier for robotics research, demanding advances in real-time decision-making, dynamic control, and multi-agent collaboration. Initiatives such as RoboCup [4] have driven progress in robot football, enabling humanoid and quadruped platforms to compete in full-scale matches [21, 5, 23, 24], while other systems have demonstrated human-level performance in table tennis [6], drone racing [25], and beyond. Yet these benchmarks tackle only isolated aspects—team tactics, split-second precision, or high-speed maneuvering—whereas the 3v3 multi-drone volleyball task combines all of these under a turn-based game structure that existing methods cannot fully address. Hierarchical control architectures offer a promising solution by managing complexity across levels of abstraction: traditional behavior-based approaches [26] and recent learning-based frameworks such as HiREPS [27] and SayCan [28] show how decomposing tasks into low-level skills and high-level planners improves adaptability and sample efficiency. Building on these insights, our work introduces an event-driven hierarchical policy that decomposes the volleyball task into a centralized high-level strategy for team tactics and multiple drone-specific low-level skills for motion control, enabling both rapid aerial agility and coordinated tactics.

6 Conclusion

To solve the 3v3 multi-drone volleyball task, we propose Hierarchical Co-Self-Play (HCSP), a novel hierarchical reinforcement learning framework. By decomposing policy into an event-driven high-level strategy for sparse strategic decisions and a suite of low-level skills for dense motion control, HCSP separates the challenges of team tactics, fine-grained control, and high-speed aerial agility. Our three-stage, demonstration-free training pipeline enables tactical strategies and agile controllers to emerge from scratch. Empirical results demonstrate that HCSP significantly outperforms baselines, and the co-self-play stage even triggers emergent behaviors without additional reward design.

Limitations

Our approach is currently limited to high-fidelity simulation, which cannot fully capture real-world sensor noise or hardware latency. Consequently, HCSP-trained policies may encounter a sim-to-real gap and are unlikely to transfer zero-shot to physical drones. Nonetheless, we report preliminary sim-to-real results in Appendix G, which provide promising initial validation in real-world experiments. Furthermore, HCSP assumes access to full state information (drone poses, velocities, and ball dynamics) rather than raw sensor or vision inputs, restricting its applicability in settings where only onboard cameras are available.

Acknowledgments

We sincerely thank Jiayu Chen, Chuqi Wang, and Yinuo Chen for their insightful discussions and experimental assistance throughout the course of this work. We are also grateful to Sicheng He for his initial involvement in the project. Their support and thoughtful input have contributed to clarifying our ideas and enhancing the overall quality of the paper.

This research was supported by National Natural Science Foundation of China (No.62406159, 62325405), Postdoctoral Fellowship Program of CPSF under Grant Number (GZC20240830, 2024M761676), China Postdoctoral Science Special Foundation 2024T170496. Additional support was provided by Tsinghua-Efort Joint Research Center for EAI Computation and Perception, Beijing National Research Center for Information Science and Technology (BNRist), Beijing Innovation Center for Future Chips, and State Key laboratory of Space Network and Communications.

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, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[2] M. Moravčík, M. Schmid, N. Burch, V. Lisỳ, D. Morrill, N. Bard, T. Davis, K. Waugh, M. Johanson, and M. Bowling. Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. *Science*, 356(6337):508–513, 2017.

[3] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[4] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa. Robocup: The robot world cup initiative. In *Proceedings of the First International Conference on Autonomous Agents*, pages 340–347, 1997.

[5] S. Liu, G. Lever, Z. Wang, J. Merel, S. A. Eslami, D. Hennes, W. M. Czarnecki, Y. Tassa, S. Omidshafiei, A. Abdolmaleki, et al. From motor control to team play in simulated humanoid football. *Science Robotics*, 7(69):eabo0235, 2022.

[6] D. B. D'Ambrosio, S. Abeyruwan, L. Graesser, A. Iscen, H. B. Amor, A. Bewley, B. J. Reed, K. Reymann, L. Takayama, Y. Tassa, et al. Achieving human level competitive robot table tennis. *arXiv preprint arXiv:2408.03906*, 2024.

[7] H. Ma, J. Fan, H. Xu, and Q. Wang. Mastering table tennis with hierarchy: a reinforcement learning approach with progressive self-play training. *Applied Intelligence*, 55(562), 2025.

[8] J. Chen, C. Yu, G. Li, W. Tang, S. Ji, X. Yang, B. Xu, H. Yang, and Y. Wang. Online planning for multi-uav pursuit-evasion in unknown environments using deep reinforcement learning. *IEEE Robotics and Automation Letters*, 10(8):8196–8203, 2025.

[9] Z. Xu, C. Yu, R. Zhang, H. Yuan, X. Yi, S. Ji, C. Wang, W. Tang, and Y. Wang. Volleybots: A testbed for multi-drone volleyball game combining motion control and strategic play. *arXiv preprint arXiv:2502.01932*, 2025.

[10] T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, J. Peters, et al. An algorithmic perspective on imitation learning. *Foundations and Trends® in Robotics*, 7(1-2):1–179, 2018.

[11] B. Xu, F. Gao, C. Yu, R. Zhang, Y. Wu, and Y. Wang. Omnidrones: An efficient and flexible platform for reinforcement learning in drone control. *IEEE Robotics and Automation Letters*, 9(3):2838–2844, 2024.

[12] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, et al. Isaac gym: High performance gpu-based physics simulation for robot learning. *arXiv preprint arXiv:2108.10470*, 2021.

[13] M. Mittal, C. Yu, Q. Yu, J. Liu, N. Rudin, D. Hoeller, J. L. Yuan, R. Singh, Y. Guo, H. Mazhar, et al. Orbit: A unified simulation framework for interactive robot learning environments. *IEEE Robotics and Automation Letters*, 8(6):3740–3747, 2023.

[14] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart. Rotors—a modular gazebo mav simulator framework. *Robot Operating System (ROS) The Complete Reference (Volume 1)*, pages 595–625, 2016.

[15] L. S. Shapley. Stochastic games. *Proceedings of the National Academy of Sciences*, 39(10): 1095–1100, 1953.

[16] M. L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Machine Learning Proceedings 1994*, pages 157–163. Elsevier, 1994.

[17] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[18] M. Lanctot, V. Zambaldi, A. Gruslys, A. Lazaridou, K. Tuyls, J. Perolat, D. Silver, and T. Graepel. A unified game-theoretic approach to multiagent reinforcement learning. *Advances in Neural Information Processing Systems*, 30, 2017.

[19] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, 1959.

[20] J. Heinrich, M. Lanctot, and D. Silver. Fictitious self-play in extensive-form games. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37, pages 805–813. PMLR, 2015.

[21] S. Liu, G. Lever, N. Heess, J. Merel, S. Tunyasuvunakool, and T. Graepel. Emergent coordination through competition. In *International Conference on Learning Representations*, 2019.

[22] A. E. Elo and S. Sloan. The rating of chessplayers: Past and present. 1978.

[23] T. Haarnoja, B. Moran, G. Lever, S. H. Huang, D. Tirumala, J. Humplik, M. Wulfmeier, S. Tunyasuvunakool, N. Y. Siegel, R. Hafner, et al. Learning agile soccer skills for a bipedal robot with deep reinforcement learning. *Science Robotics*, 9(89):eadi8022, 2024.

[24] Z. Xiong, B. Chen, S. Huang, W.-W. Tu, Z. He, and Y. Gao. Mqe: Unleashing the power of interaction with multi-agent quadruped environment. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5918–5924. IEEE, 2024.

[25] E. Kaufmann, L. Bauersfeld, A. Loquercio, M. Müller, V. Koltun, and D. Scaramuzza. Champion-level drone racing using deep reinforcement learning. *Nature*, 620(7976):982–987, 2023.

[26] R. C. Arkin. *Behavior-based robotics*. MIT press, 1998.

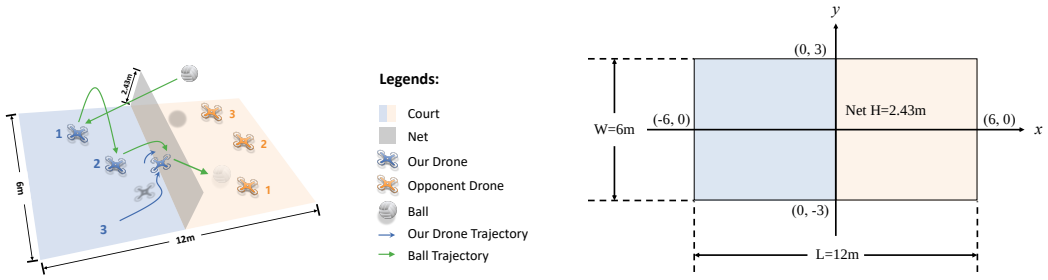
[27] C. Daniel, G. Neumann, O. Kroemer, and J. Peters. Hierarchical relative entropy policy search. *Journal of Machine Learning Research*, 17(93):1–50, 2016.

[28] A. Brohan, Y. Chebotar, C. Finn, K. Hausman, A. Herzog, D. Ho, J. Ibarz, A. Irpan, E. Jang, R. Julian, et al. Do as i can, not as i say: Grounding language in robotic affordances. In *Proceedings of The 6th Conference on Robot Learning*, pages 287–318. PMLR, 2023.

[29] C. Yu, A. Velu, E. Vinitsky, J. Gao, Y. Wang, A. Bayen, and Y. Wu. The surprising effectiveness of ppo in cooperative multi-agent games. *Advances in Neural Information Processing Systems*, 35, 2022.

[30] R. Zhang, Z. Xu, C. Ma, C. Yu, W.-W. Tu, W. Tang, S. Huang, D. Ye, W. Ding, Y. Yang, et al. A survey on self-play methods in reinforcement learning. *arXiv preprint arXiv:2408.01072*, 2024.

A Details of the 3v3 Muiti-Drone Volleyball Task



(a) Task overview.

(b) Top-down view of the court layout.

Figure 7: Illustrations of the 3v3 multi-drone volleyball task.

A.1 Court

The 3v3 multi-drone volleyball task is illustrated in Fig. 7a, with a top-down, 2D court layout shown in Fig. 7b. The origin lies at the center of the court, and the y-axis divides it into two equal halves (each team's side). The x-axis runs the 12m length (from $x = -6$ to $x = 6$), while the court spans 6m in width along the y-axis (from $y = -3$ to $y = 3$). A net of height 2.43m stretches horizontally along the line $x = 0$, from $(0, -3)$ to $(0, 3)$.

A.2 Drone

Our primary drone platform is the *Iris* quadrotor [14], virtually equipped with a 0.2 m radius racket (coefficient of restitution 0.8) for ball striking. The drone's root state used by the simulation framework is described by a 23-dimensional vector containing its position, rotation (quaternion), linear and angular velocities, forward and upward orientation vectors (corresponding to the first two columns of the rotation matrix derived from the quaternion), and normalized rotor speeds. The control dynamics governing the drone's motion arise from its physical configuration and the interplay of forces and torques, as described below:

$$\dot{\boldsymbol{x}}_W = \boldsymbol{v}_W, \quad \dot{\boldsymbol{v}}_W = \boldsymbol{R}_{WB} f + \boldsymbol{g} + \boldsymbol{F} \tag{3}$$

$$\dot{\boldsymbol{q}} = \frac{1}{2}\boldsymbol{q} \otimes \boldsymbol{\omega}, \quad \dot{\boldsymbol{\omega}} = \boldsymbol{J}^{-1}(\boldsymbol{\eta} - \boldsymbol{\omega} \times \boldsymbol{J}\boldsymbol{\omega}) \tag{4}$$

where $\boldsymbol{x}_W$ and $\boldsymbol{v}_W$ are the position and velocity of the drone in the world frame, $\boldsymbol{R}_{WB}$ is the rotation matrix from the body frame to the world frame, $\boldsymbol{J}$ is the diagonal inertia matrix, $\boldsymbol{g}$ represents gravity, $\boldsymbol{q}$ is attitude in quaternion, and $\boldsymbol{\omega}$ is the angular velocity. The operator for quaternion multiplication is denoted by $\otimes$. External forces $\boldsymbol{F}$ include aerodynamic drag and downwash effects. The collective thrust $\boldsymbol{f}$ and torque $\boldsymbol{\eta}$ are calculated based on the thrust of each rotor $\boldsymbol{f}_i$ as:

$$\boldsymbol{f} = \sum_i \boldsymbol{R}_B^{(i)} \boldsymbol{f}_i \tag{5}$$

$$\boldsymbol{\eta} = \sum_i \boldsymbol{T}_B^{(i)} \times \boldsymbol{f}_i + sign_i k_i \boldsymbol{f}_i \tag{6}$$

where $\boldsymbol{R}_B^{(i)}$ and $\boldsymbol{T}_B^{(i)}$ are the orientation and position of the i-th rotor in the body frame, k_i is the force-to-moment ratio and $sign_i$ is 1 for clockwise propellers and -1 for counterclockwise propellers.

A.3 Game Rules

The multi-drone game is governed by a set of well-defined rules that closely mimic the cooperative and competitive dynamics of real-world volleyball, subject to the following simplifications. Unlike standard volleyball, there is no positional rotation or fixed court zones, so drones do not change roles between rallies, and the three-meter attack line is omitted. At the start of each episode, a serving side is randomly selected, and the ball is released in free fall from a point 5m above the midpoint of that team's baseline, with slight random positional noise.

During the rally phase, teams alternate in attempting to return the ball over the net while obeying two key constraints: (i) each team is allowed no more than three successive hits before the ball must cross to the opponent's side, (ii) the same drone is not permitted to hit the ball twice in succession.

The game outcome is determined by a series of terminal conditions that reflect violations of basic physical or tactical rules. A team loses a rally if (i) any of its drones hits the ground (i.e., descends below a predefined altitude threshold), (ii) any drone illegally crosses the net, (iii) any drone performs a racket hit in violation of timing or rule constraints, (iv) the ball lands on their own side of the court, (v) the team hit the ball outside the boundary of the court, or (vi) the ball collides with the net due to their last action.

In rare cases, both sides violate the rules of the game at the same time. If drones of both sides hit the ground or both sides cross the net in the same simulation step, a draw will be declared. This ensures that the environment maintains fairness while accounting for simultaneity in physical interactions.

B Details of Stage I: Low-Level Skill Acquisition

Ensuring smooth transitions between temporally adjacent low-level primitives—especially when control must switch between different drones or skill types—poses a significant challenge. To address this, we employ policy chaining, iteratively training each primitive so it executes seamlessly under the event-driven high-level strategy. Concretely, we implement seven fundamental low-level skills (*serve, receive, set, attack, pass, hover, goto*) as single-agent reinforcement learning (RL) tasks, each of which is trained separately using Proximal Policy Optimization (PPO) [17].

Moreover, for ball-interaction skills (*serve, receive, set, attack, pass*), we design a three-stage reward: (i) The before hit phase: optimal preparation incentives through alignment rewards and posture optimization; (ii) The after hit phase: contact quality evaluation based on ball dynamics metrics; (iii) The end phase: bonuses for task success or penalties for incorrect end states.

B.1 Low-Level Skills

B.1.1 Serve

Description. The *serve* skill is executed by the serving drone of the randomly selected team at the start of each episode. Given a designated target point on the opponent's court, the drone must hit the ball once to send it close to that location. The skill's duration spans from the beginning of the episode to the moment the serving drone makes contact with the ball.

Observation and reward. The drone's observation is a vector of dimension 37 including the drone's root state, the ball's position, the ball's relative position to the drone, the ball's linear velocity, a one-hot flag indicating whether the drone is permitted to hit the ball, and the ball's relative position to the target ball position. The detailed specification of the task's reward function per time step can be found in Table 3.

B.1.2 Receive

Description. The *receive* skill is used by the receiving drone to counter the opponent's serve. During the *receive* skill training, the opponent's serving drone executes the *serve* skill as part of the environment. The receiving drone needs to hit the ball to the passing drone positioned at the front-

Table 3: Reward of the *serve* skill

Type	Name	Sparse	Value Range	Description
Before Hit	dist_to_ball	✗	$[0, 0.16]$	drone's distance to the ball
	drone_hit_ball	✓	$\{0, 10\}$	drone hitting the ball
	penalty_pos_x	✓	$\{-150, 0\}$	being too close to the net
	penalty_pos_z	✗	$[-\infty, 0]$	being too high or too low
	penalty_roll	✗	$[-\infty, 0]$	excessive roll angle
	penalty_yaw	✗	$[-\infty, 0]$	excessive yaw angle
After Hit	dist_to_anchor	✗	$[0, 16]$	ball's distance to anchor
End Reward	in_side	✓	$\{0, 10\}$	ball landing in opponent's court
	highest_ball_pos	✓	$\{0, 1.5\}$	ball's peak height exceeding 3m
	penalty_ground_collision	✓	$\{-0.1, 0\}$	drone colliding with the ground
	penalty_wrong_hit	✓	$\{-10, 0\}$	drone not using racket to hit the ball

Table 4: Reward of the *receive* skill

Type	Name	Sparse	Value Range	Description
Before Hit	dist_to_ball	✗	$[0, 10]$	drone's distance to the ball
	drone_hit_ball	✓	$\{0, 150\}$	drone hitting the ball
	penalty_pos_x	✓	$\{-5, 0\}$	being too close to the net
	penalty_pos_z	✗	$[-\infty, 0]$	being too high or too low
	penalty_roll	✗	$[-\infty, 0]$	excessive roll angle
	penalty_yaw	✗	$[-\infty, 0]$	excessive yaw angle
After Hit	dist_to_anchor	✗	$[0, 50]$	ball's distance to anchor
	ball_vel_direction	✗	$[0, 150]$	velocity direction to the anchor
	secondary_dist_to_anchor	✗	$[0, 200]$	height-limited ball's distance to anchor
End Reward	in_side	✓	$\{0, 10\}$	ball landing in opponent's court
	highest_ball_pos	✗	$[0, 200]$	ball's peak height exceeding 3m
	penalty_ground_collision	✓	$\{-100, 0\}$	drone colliding with the ground
	penalty_wrong_hit	✓	$\{-10, 0\}$	drone not using racket to hit the ball

left side of its own court. The skill's duration spans from the moment the opponent's serving drone hits the ball to the moment the receiving drone makes contact with it.

Observation and reward. The drone's observation is a vector of dimension 34 including the drone's root state, the ball's position, the ball's relative position to the drone, the ball's linear velocity, and a one-hot flag indicating whether the drone is permitted to hit the ball. The detailed specification of the task's reward function per time step can be found in Table 4.

B.1.3 Set

Description. The *set* skill is executed by the setting drone and constitutes the second touch during a team's turn. During the *set* skill training, the ball is initialized using a Gaussian distribution to simulate the moment it is passed by the teammate. The setting drone is required to redirect the ball to the front-right side of its own court, enabling the attacking drone to perform a strike. The skill's duration spans from the moment the passing drone contacts the ball to the moment the setting drone hits it.

Observation and reward. The drone's observation is a vector of dimension 34 including the drone's root state, the ball's position, the ball's relative position to the drone, the ball's linear velocity, and a one-hot flag indicating whether the drone is permitted to hit the ball. The detailed specification of the task's reward function per time step can be found in Table 5.

Table 5: Reward of the *set* skill

Type	Name	Sparse	Value Range	Description
Before Hit	dist_to_ball	✗	$[0, 2]$	drone's distance to the ball
	drone_hit_ball	✓	$\{0, 40\}$	drone hitting the ball
	penalty_pos_xy	✗	$[-\infty, 0]$	being too far from the ideal point
	penalty_pos_z	✗	$[-\infty, 0]$	being too high or too low
	penalty_yaw	✗	$[-\infty, 0]$	excessive yaw angle
After Hit	dist_to_anchor	✗	$[0, 10]$	ball's distance to anchor
	drone_dist_to_anchor	✗	$[0, 200]$	drone's distance to anchor
End Reward	highest_ball_pos	✗	$[0, 400]$	ball's peak height exceeding 3m
	penalty_ground_collision	✓	$\{-10, 0\}$	drone colliding with the ground
	penalty_wrong_hit	✓	$\{-10, 0\}$	drone not using racket to hit the ball

Table 6: Reward of the *attack* skill

Type	Name	Sparse	Value Range	Description
Before Hit	dist_to_ball	✗	$[0, 1]$	drone's distance to the ball
	drone_hit_ball	✓	$\{0, 20\}$	drone hitting the ball
	drone_pos_z	✗	$[0, 400]$	height of drone hitting the ball
	penalty_yaw	✗	$[-2\pi, 0]$	excessive yaw angle
	penalty_roll	✗	$[-2\pi, 0]$	excessive roll angle
After Hit	dist_to_anchor	✗	$[0, 150]$	ball's distance to anchor
	ball_vel_x	✓	$\{0, 1\}$	velocity direction of the ball
	ball_vel_z	✓	$\{0, 150\}$	downward velocity of the ball
End Reward	ball_hit_ground	✓	$\{0, 5\}$	ball hit the ground successfully
	ball_final_vel	✗	$[0, \infty]$	ball's velocity hitting the ground
	in_side	✓	$\{0, 10\}$	ball landing in opponent's court
	penalty_ground_collision	✓	$\{-10, 0\}$	drone colliding with the ground
	penalty_wrong_hit	✓	$\{-10, 0\}$	drone not using racket to hit the ball
	penalty_hit_net	✓	$\{-10, 0\}$	ball hit the net

B.1.4 Attack

Description. The *attack* skill is executed by the attacking drone and constitutes the third touch in a team's sequence. During the *attack* skill training, the setting drone performs the *set* skill as part of the environment. The attacking drone must deliver an effective offensive strike toward one of two predefined target positions on the opponent's court (left or right). The duration of the skill spans from the moment the setting drone contacts the ball to the moment the attacking drone strikes it.

Observation and reward. The drone's observation is a vector of dimension 36 including the drone's root state, the ball's position, the ball's relative position to the drone, the ball's linear velocity, a one-hot flag indicating whether the drone is permitted to hit the ball, and a one-hot flag indicating the attack direction. The detailed specification of the task's reward function per time step can be found in Table 6.

B.1.5 Pass

Description. The *pass* skill is used to counter the opponent's attack and constitutes the first touch in a team's sequence. During the *pass* skill training, the opponent side includes a setting drone that performs the *set* skill and an attacking drone that performs the *attack* skill as part of the environment. The passing drone needs to hit the ball to the teammate positioned at the front-left side of its own court. The duration of the skill spans from the moment the opponent's attacking drone spikes the ball to the moment the passing drone makes contact with it.

Table 7: Reward of the *pass* skill

Type	Name	Sparse	Value Range	Description
Before Hit	dist_to_ball	✗	$[0, 10]$	drone's distance to the ball
	drone_hit_ball	✓	$\{0, 150\}$	drone hitting the ball
	drone_vel_z	✗	$[0, 2.5]$	drone's rising speed
	penalty_pos_x	✓	$\{-5, 0\}$	being too close to the net
	penalty_pos_z	✗	$[-\infty, 0]$	being too high or too low
	penalty_roll	✗	$[-0.5\pi, 0]$	excessive roll angle
	penalty_yaw	✗	$[-0.5\pi, 0]$	excessive yaw angle
After Hit	dist_to_anchor	✗	$[0, 10]$	ball's distance to anchor
	ball_vel_direction	✗	$[0, 150]$	velocity direction to the anchor
	secondary_dist_to_anchor	✗	$[0, 200]$	height-limited ball's distance to anchor
End Reward	highest_ball_pos	✗	$[0, 200]$	ball's peak height exceeding 3m
	penalty_ground_collision	✓	$\{-100, 0\}$	drone colliding with the ground
	penalty_not_hit_ball	✓	$\{-10, 0\}$	drone not hitting the ball
	penalty_wrong_hit	✓	$\{-10, 0\}$	drone not using racket to hit the ball

Table 8: Reward of the *hover* skill

Type	Name	Sparse	Value Range	Description
Hover Reward	drone_pos	✗	$[0, 3]$	drone's relative position to the target point
	drone_up	✗	$[0, 3]$	drone upright position
	drone_spin	✗	$[0, 3]$	drone spin suppression

Observation and reward. The drone's observation is a vector of dimension 36 including the drone's root state, the ball's position, the ball's relative position to the drone, the ball's linear velocity, a one-hot flag indicating whether the drone is permitted to hit the ball, and a one-hot flag indicating the opponent's attacking direction. The detailed specification of the task's reward function per time step can be found in Table 7.

B.1.6 Hover

Description. The *hover* skill is executed immediately after the drone hits the ball. The drone stabilizes at a predetermined altitude and position to prepare for the next move. Given that different ball-interaction skills terminate in distinct end states, we define five corresponding hover skills (*serve_hover, receive_hover, set_hover, attack_hover, pass_hover*) and train each independently to stabilize the agent after the associated interaction with the same hover reward. Here, we specifically illustrate the *hover* skill following the *serve* skill. During training, the serving drone performs the *serve* skill as part of the environment, and immediately after it contacts the ball, its policy switches to the *hover* skill.

Observation and reward. The drone's observation is a vector of dimension 26 including the drone's relative position to the target point, the drone's root state except position, and the drone's relative orientation to the ideal state. The detailed specification of the task's reward function per time step can be found in Table 8.

B.1.7 GoTo

Description. The *goto* skill is performed after the drone has stabilized and involves navigating to a designated target position at a fixed altitude, preparing the drone for an upcoming hit. In training, the drone's initial and target positions are randomized to encourage generalization across various scenarios.

Observation and reward. The drone's observation is a vector of dimension 26 including the drone's relative position to the target point, the drone's root state except position, and the drone's relative

Table 9: Reward of the *goto* skill

Type	Name	Sparse	Value Range	Description
Hover Reward	drone_pos	✗	$[0, 1]$	drone's relative position to the target point
	drone_up	✗	$[0, 1]$	drone upright position
	drone_spin	✗	$[0, 1]$	drone spin suppression
	drone_effort	✗	$[0, 0.1]$	hover with low effort

orientation to the ideal state. The detailed specification of the task's reward function per time step can be found in Table 9.

B.2 Training Result

The training results of low-level skills are shown in Fig. 8.

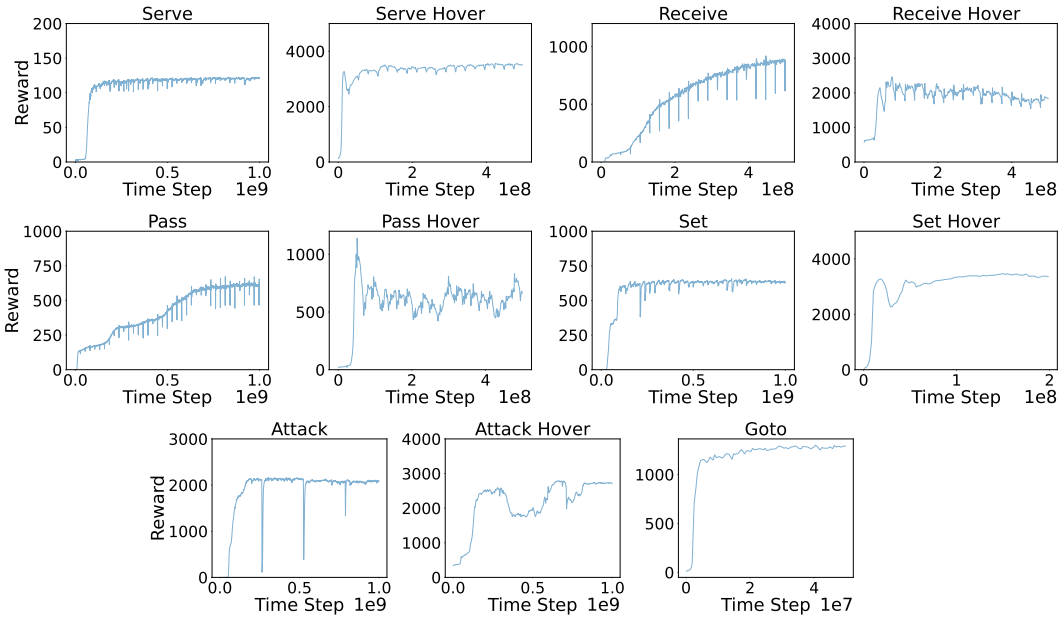
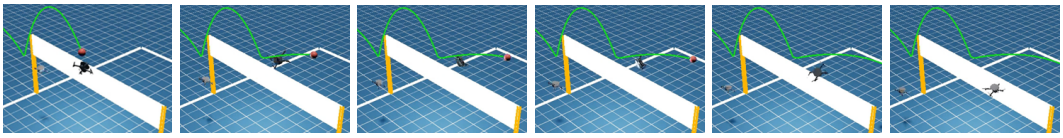


Figure 8: The training results of low-level skills.

B.3 Case Study

When training the *attack* and *attack_hover* skills through policy chaining, we observed the emergence of a front-flip attack (Fig. 9). Executing a front flip is particularly challenging for a drone, since it requires precise timing, rapid attitude changes, and tight control over thrust to complete a full somersault without losing stability. Remarkably, even without explicitly designing for this maneuver, policy chaining can facilitate learning to perform a front flip to attack the ball. Moreover, performing a front-flip attack allows the drone to impart additional downward and forward momentum at the moment of impact, resulting in a higher ball speed. The finding shows that policy chaining not only facilitates complex skill discovery but also naturally promotes the emergence of aggressive, high-performance behaviors through the composition of simpler skills.



(a) (b) (c) (d) (e) (f)

Figure 9: Sequence of six frames, sampled sequentially in time from the start to the completion of the front-flip attack.

B.4 Hyperparameters

The PPO [17] hyperparameters applied to all low-level skill training tasks are listed in Table 10. All parameters in the table are used unchanged except for the actor and critic learning rates:

- *receive*, *pass*, and *pass_hover*: actor learning rate & critic learning rate = 1×10^{-5}
- *receive_hover*: actor learning rate & critic learning rate = 1×10^{-4}

Table 10: Hyperparameters used for PPO in low-level skill training.

hyperparameters	value	hyperparameters	value	hyperparameters	value
optimizer	Adam	max grad norm	10	entropy coef	0.001
buffer length	64	num minibatches	16	PPO epochs	4
value norm	ValueNorm1	clip param	0.1	normalize advantages	True
actor learning rate	5×10^{-4}	hidden sizes	[256, 128, 128]	GAE lambda	0.95
critic learning rate	5×10^{-4}	gain	0.01	GAE gamma	0.995
max episode length	500	num envs	4096	train steps	1×10^{9}

C Details of Stage II: High-Level Strategy Pretraining

The high-level strategy is invoked only at discrete events, namely when the ball is struck or crosses the net, while low-level skills execute continuously between those moments. In Stage II, we freeze the low-level skills and train the high-level strategy using $\text{PSRO}_{\text{Nash}}$ [18], an iterative game-theoretic self-play method where policies compete against opponents sampled from a Nash equilibrium distribution. To address the sparsity of high-level decisions, we employ sample reallocation, focusing training on event-triggered timesteps and redistributing rewards across intervals. The high-level strategy is implemented as a centralized multi-layer perceptron (MLP) with three output heads, each issuing commands to a specific drone, thereby enabling coordinated team-level decision making. The entire framework is trained using MAPPO [29].

C.1 High-Level Strategy

Observation. The high-level strategy's observation is a vector of dimension 100 including the root states (position, rotation, linear velocity, angular velocity) of 6 drone agents (3 on each side), the ball's position, the ball's linear velocity, binary flags indicating which drone agents have already hit the ball, ball side information, a one-hot representation of the current game phase.

Action. The high-level action space is organized as a multi-head structure with three heads of sizes 6, 3, and 4, corresponding to the passing, setting, and attacking drone roles. Each head defines an independent categorical distribution: its output logits are converted to normalized action probabilities via a softmax activation. Prior to discretizations, the logits for each head are sampled from Gaussian distributions in a latent policy space, thereby facilitating probabilistic exploration.

Reward. The reward for the high-level strategy π_j^H at timestep t is defined as:

$$r_{j,t}^H = c_1 \times \text{win\_or\_lose}_j + c_2 \times \text{racket\_hit\_ball}_j, \tag{7}$$

where $\text{win\_or\_lose}_j = 1$ if team j wins, -1 if it loses, and 0 otherwise; $\text{racket\_hit\_ball}_j = 1$ if any racket of team j contacts the ball at timestep t, and 0 otherwise.

C.2 Training Result

The results of high-level strategy training are shown in Fig. 10. In our experiments, we set the maximum population size to five, yielding five iterative training phases; each phase terminates when either a preset number of training steps is reached or a win-rate threshold is met. The lower triangular portion of the matrix reports the win rate of each row policy against each column policy (i.e., the probability that the row policy prevails). Because the game is symmetric and zero-sum, the upper-triangular entries are complementary, such that each pair of win rates sums to one.

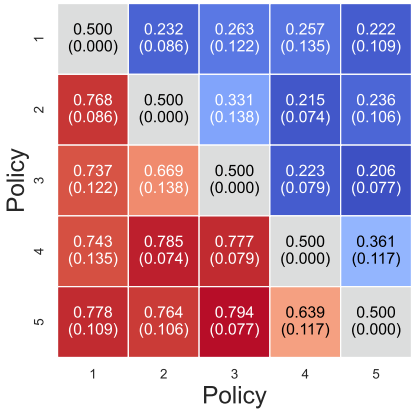


Figure 10: Win-rate heatmap illustrating the evolution of high-level strategy training in Stage II.

C.3 Hyperparameters

The hyperparameters used for $\text{PSRO}_{\text{Nash}}$ in high-level strategy training are shown in Table 11.

Table 11: Hyperparameters used for $\text{PSRO}_{\text{Nash}}$ in high-level strategy training.

hyperparameters	value	hyperparameters	value	hyperparameters	value
optimizer	Adam	max grad norm	10	entropy coef	0.001
buffer length	64	num minibatches	16	PPO epochs	4
value norm	ValueNorm1	clip param	0.1	normalize advantages	True
actor learning rate	1×10^{-4}	hidden sizes	$[256, 128, 128]$	GAE lambda	0.95
critic learning rate	1×10^{-4}	gain	0.01	GAE gamma	0.995
max episode length	1500	num envs	512	min iteration steps	1.6384×10^6
max population size	5	win rate threshold	0.9	max iteration steps	1.6384×10^8

D Details of Stage III: Co-Self-Play

Stage III consists of two steps (Fig. 3): the low-level skill refinement step and the high-level strategy relearning step. In the low-level skill refinement step, the high-level strategy together with every low-level skill except *serve* and *receive* are co-optimized, resulting in updated low-level skills, which are then added to the existing low-level skill pool. The previous high-level strategy becomes invalid due to the change in the size of the skill pool. Therefore, the high-level strategy relearning step involves retraining the high-level strategy to accommodate the augmented low-level skill pool. To analyze the low-level skill refinement step, we conduct two ablation studies: one on the refinement schedule (Appendix D.1) and the other on the KL divergence penalty term (Appendix D.2). For the

high-level strategy relearning step, we perform an ablation study on the initialization of high-level strategy parameters (Appendix D.3).

D.1 Ablation Study on Low-Level Skill Refinement Schedule

Instead of jointly co-optimizing the high-level strategy with all low-level skills simultaneously, we co-optimize the high-level strategy with one low-level skill at a time (Algo. 1). We compare the two co-optimization methods by evaluating the win rates of the saved checkpoints against the high-level strategy and low-level skills obtained after Stage II, under identical settings. These settings include the reward structure, maximum number of iterations ($M = 4000$), checkpoint saving interval ($N = 1000$), and win rate threshold ($t = 0.7$). Each iteration collects 32768 sampled steps. All results are averaged over three random seeds. As shown in Fig. 11a, jointly co-optimizing the high-level strategy with all low-level skills leads to near-zero win rates. In contrast, co-optimizing the high-level strategy with one low-level skill at a time can yield checkpoints that achieve win rates above 50%. Since MARL is inherently non-stationary, simultaneously updating all low-level skills destabilizes the learning process. In comparison, optimizing one low-level skill at a time reduces non-stationarity and enables more stable and effective learning.

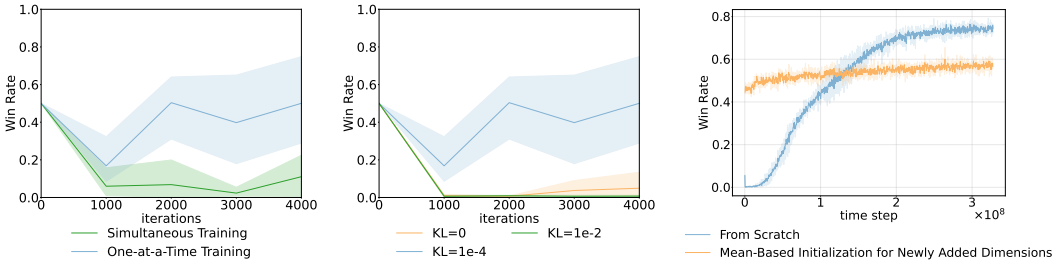
Algorithm 1 High-level strategy and one-at-a-time low-level skill joint optimization.

Require: Low-level skill set $\{\pi^L\}$, high-level policy π^H, number of max iterations M, checkpoint saving interval N, win rate threshold τ
Ensure: Augmented low-level skill set $\{\pi^L_{\text{aug}}\}$

Initialize $\Pi_{\text{new}} \leftarrow \emptyset$
for each $\pi^L_k \in \{\pi^L\}$ **do**
  Freeze parameters of all other low-level policies in $\{\pi^L\}$
  $\pi^{L'}_k \leftarrow \pi^L_k$
  $\pi^{H'}_k \leftarrow \pi^H$
  Form team $(\{\pi^L\} \setminus \{\pi^L_k\}) \cup \{\pi^{L'}_k\}$ with high-level $\pi^{H'}_k$
  for $i = 1$ to M **do**
    Jointly optimize $\pi^{L'}_k$ and $\pi^{H'}_k$ through self-play against the frozen team $\{\pi^L\}$ and π^H
    if $i \bmod N = 0$ **then**
      Save a checkpoint of $\pi^{L'}_k$ and $\pi^{H'}_k$
    end if
  end for
  For each $\pi^{L'}_k, \pi^{H'}_k$ checkpoint, form team $(\{\pi^L\} \setminus \{\pi^L_k\}) \cup \{\pi^{L'}_k\}$ with high-level $\pi^{H'}_k$ and evaluate the win rate against the frozen team $\{\pi^L\}$ and π^H.
  Select the checkpoint of $\pi^{L'}_k$ with the highest win rate, denoted as $\pi^L_{k,\text{best}}$
  if win rate of $\pi^L_{k,\text{best}} > \tau$ **then**
    Append $\pi^L_{k,\text{best}}$ to Π_{new}
  end if
end for
return $\{\pi^L\} \cup \Pi_{\text{new}}$

D.2 Ablation Study on KL Divergence Penalty

As stated in Sec. 3.3, we incorporate a KL divergence penalty term into the low-level skill reward, in addition to the game-result-related reward shared with the high-level strategy. Here, we conduct a detailed ablation study on the KL divergence penalty to investigate its influence on training performance. As shown in Fig. 11b, and following Algo. 1, we compare different settings of the KL penalty by evaluating the win rates of the saved checkpoints against the high-level strategy and low-level skills obtained after Stage II. We observe that a large KL penalty coefficient (e.g., $1\text{e}{-2}$) causes the policy performance to collapse rapidly; conversely, without a KL penalty (coefficient = 0), the policy hovers only slightly above zero win rate. Only with a moderate coefficient (e.g., $1\text{e}{-4}$) does the policy yield checkpoints achieving win rates above 50%. This is because, without a KL penalty,



(a) Comparison of win rates under different co-optimization training schedules.

(b) Comparison of win rates under different KL divergence coefficients.

(c) Comparison of win rates under different high-level strategy parameter initializations.

Figure 11: Ablation study results in Stage III.

the reward signal for low-level strategies becomes sparse, significantly slowing down learning. On the other hand, an overly large KL penalty forces the policy to remain too close to the original one, which leads to accumulated error over time as the policy rigidly imitates previous behaviors without adaptation.

D.3 Ablation Study on High-Level Strategy Parameter Initialization

As the previous high-level strategy becomes invalid due to changes in the size of the low-level skill pool, it is necessary to retrain the high-level strategy to accommodate the augmented skill set. The initialization of this new high-level strategy is a critical factor influencing the overall performance in Stage III. We compare two different initialization methods for the high-level strategy. The first is a straightforward approach that trains the high-level strategy from scratch. In the second approach, we expand each actor head's parameter vector to match the larger action space by copying over its original values. Any newly added dimensions are then initialized to the mean of that head's original parameters. In both settings, we train the new high-level strategy with frozen augmented low-level skills against the frozen high-level strategy and low-level skills obtained after Stage II. As shown in Fig. 11c, although training from scratch requires more time steps to converge, it ultimately achieves a higher win rate. In contrast, the parameter-averaging initialization method tends to converge to a suboptimal local solution.

E Baselines

We evaluate five baseline methods for this multi-drone game, including four non-hierarchical game-theoretic approaches and one rule-based hierarchical baseline adapted from VolleyBots [9]. The game-theoretic baselines follow estabilished literature [30] and comprise Self-Play (SP) [19], Fictitious Self-Play (FSP) [20], Policy-Space Response Oracles [18] with Nash meta-solver ($\text{PSRO}_{\text{Nash}}$), and PSRO with uniform meta-solver ($\text{PSRO}_{\text{Uniform}}$). Additionally, we incorporate a rule-based hierarchical baseline from VolleyBots.

SP establishes the foundation by training agents against their current policy versions. FSP extends this by maintaining a historical policy pool, forcing agents to adapt to an averaged opponent strategy. $\text{PSRO}_{\text{Nash}}$ employs Nash equilibrium weighting for strategic policy selection, while $\text{PSRO}_{\text{Uniform}}$ simplifies this by sampling policies uniformly at each iteration.

The rule-based hierarchical baseline from VolleyBots implements a structured volleyball-inspired gameplay pipeline. In this system, the serving drone initiates play by executing a serve to a specific target position on the opponent's court. The receiving team then follows a deterministic sequence: (i) the passing drone receives the ball and directs it to the setting drone, (ii) the setting drone positions the ball for optimal attack placement, and (iii) the attacking drone executes an offensive strike.

To ensure fairness, we provide all baseline agents with the full-state observations of opponent drones as those available to HCSP. Additionally, HCSP is trained independently without exposure to any of the baseline agents, guaranteeing an unbiased evaluation.

F Evaluation Metrics

F.1 Win Rate

The win rate is computed by evaluating two populations across 500 independent episodes. At the beginning of each episode, one strategy is sampled from each population according to a specified sampling distribution. The two strategies then compete, and the outcome (win/loss/draw) is recorded. The final win rate represents the proportion of episodes won by one population relative to the other, averaged over 3 random seeds to ensure statistical reliability.

F.2 Nash-Averaging Evaluator

While head-to-head win rates fail to reflect absolute policy strength in non-transitive games, and Elo ratings [22] exhibit sensitivity to the composition of the policy population, the Nash-averaging evaluator, introduced by Liu et al. [21], provides a robust alternative. This method selects a randomly sampled set of policies, then constructs a payoff matrix from pairwise match-ups, and finally computes a Nash equilibrium (NE). The resulting equilibrium weights yield a comparatively population-invariant measure of relative performance, mitigating biases induced by cyclic dominance. So we adopt this as an evaluation metric. The details are as follows.

To compute a Nash-averaging evaluator, we first randomly select 10 different policy checkpoints: 1 each from SP, FSP, $\text{PSRO}_{\text{Uniform}}$, and $\text{PSRO}_{\text{Nash}}$, along with 3 from Stage II and 3 from Stage III of training. All checkpoints are drawn from completely independent training runs with different random seeds. Each policy pair then competes in 500 independent episodes under identical conditions, with the row policy's win rate against the column policy recorded as the corresponding matrix element. All results are averaged across three seeds to ensure statistical reliability. Then, we get a 10×10 win rate heatmap (Fig. 4b).

The meta policy is derived by computing the NE of the 10×10 win rate matrix. Since the underlying game is zero-sum, win rates (probabilistic outcomes) must first be converted into zero-sum payoff values within the range $[-1, 1]$, where one agent's gain is the other's loss. Specifically, the conversion follows:

$$\text{payoffs}[i, j] = 2 \times \text{win\_rates}[i, j] - 1 \tag{8}$$

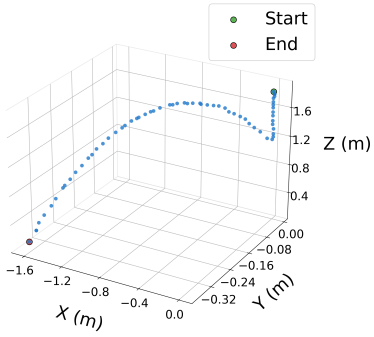
We then compute each player's NE-based optimal mixed strategy, ensuring neither player can gain a higher expected payoff by unilaterally deviating from their strategy. This computed mixed strategy is the Nash-averaging evaluator we used in Sec. 4.2.1. We then evaluate the Stage II and Stage III population against this Nash-averaging evaluator in the same way described in Appendix F.1.

G Preliminary Real-World Experiments

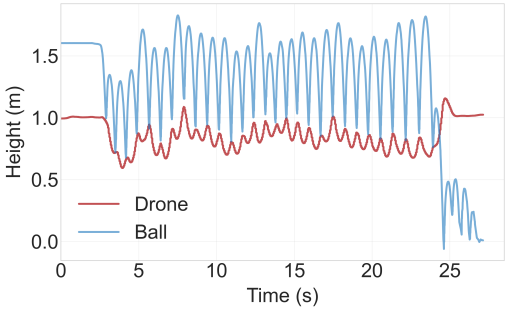
Although the current method is not yet sufficient for full 3v3 zero-shot deployment, we incorporate key real-world factors in simulation. Our preliminary Sim2Real adaptations include:

- We support two action space options. In addition to the 50Hz PRT, we also support 50Hz CTBR policy output followed by an 800Hz PID controller.
- We add both domain randomization (initial states and restitution coefficient) and observation latency.

(a) Drone setup (b) *Serve* (c) *Solo Bump*



(d) Ball trajectory in *Serve*



(e) Height variation in *Solo Bump*

Figure 12: Real-world experiments.

We further conduct preliminary real-world experiments to demonstrate the zero-shot Sim2Real potential of our approach on learned low-level skills, such as the *Serve* skill and a *Solo Bump* task, in which the drone is required to bump the ball repeatedly to showcase its agility and stability. As shown in Fig. 12a, to stay within the quadrotor's thrust margin, we mount a badminton racket (diameter 22cm, mass 60g) on top of the drone and use a small foam ball (diameter 4cm, mass 47g). Both the drone and the ball are tracked by a motion-capture system: the drone, modeled as a rigid body, receives position and orientation directly from mocap, while an Extended Kalman filter fuses these data with PX4 IMU readings to estimate velocity; the ball, treated as a point mass, also gets position from mocap, and a Kalman filter differentiates these measurements to recover velocity. The policies are deployed on the onboard Nvidia Orin processor. In zero-shot trials, the low-level skills such as *Serve* are executed successfully (Fig. 12b and 12d) and the *Solo Bump* policy achieves 29 consecutive bumps (Fig. 12c and 12e). These results provide promising evidence for the physical plausibility and transferability of our approach.