Enhancing Language Agent Strategic Reasoning through Self-Play in Adversarial Games

Anonymous ACL submission

Abstract

Existing language agents often encounter difficulties in dynamic adversarial games due to poor strategic reasoning. To mitigate this limitation, a promising approach is to allow agents to learn from game interactions automatically, without relying on costly expert-labeled data. Unlike static environments where agents receive fixed feedback or rewards, selecting appropriate opponents in dynamic adversarial games can significantly impact learning performance. However, the discussion of opponents in adversarial environments remains an area under exploration. In this paper, we propose a **S**tep-level poli**C**y **O**ptimization method through **P**lay-**A**nd-**L**earn, **SCO-PAL**. Leveraging SCO-PAL, we conduct a detailed analysis of opponent selection by setting opponents at different levels and find that self-play is the most effective way to improve strategic reasoning in such adversarial environments. Utilizing SCO-PAL with self-play, we increase the average win rate against four opponents by approximately 30% compared to baselines and achieve a 54.76% win rate against GPT-4 in six adversarial games.

1 Introduction

Large Language Model (LLM)–based agents have achieved remarkable success in a variety of language-centric tasks—including web navigation (Zhou et al., 2023; Deng et al., 2024; Putta et al., 2024; Iong et al., 2024), embodied interactions (Wang et al., 2023; Zhu et al., 2023; Lin et al., 2024) and tool invocation (Qin et al., 2023; Shen et al., 2024). Unlike classic RL agents that operate over a small, well-defined set of discrete or continuous controls and clear state transitions, LLM agents must generate high-dimensional natural-language actions conditioned on long, unstructured textual contexts. This fundamentally complicates strategic reasoning in adversarial games, where success hinges on planning over many moves and reacting to an opponent's evolving strategy under sparse, delayed rewards. Benchmarks such as `GTBench` (Duan et al., 2024) (e.g. `Breakthrough`, `Connect Four`, `Nim`) expose these challenges by testing both the depth of planning and the fluency of language-based decision making.

To mitigate the shortcomings in the strategic reasoning of language agents, there are mainly two paths. One is imitating a strong strategy directly inspired by Chen et al. (2023); Zeng et al. (2023). This way is simple and effective, but gathering expert-level data is time-consuming, costly, and challenging to scale. Therefore, studies on another path focus on **play-and learn** paradigm, which enriches data through interaction and learning from the interactions. For example, Song et al. (2024); Xiong et al. (2024); Xi et al. (2024) allow agents to interact autonomously with environments, gathering numerous data for improvement. However, unlike interacting with unchanging conditions in static environments, the agent in adversarial games faces dynamic and diverse opponents, which brings additional challenges. In adversarial scenarios, Wang et al. (2024b); Cheng et al. (2024) enhance the strategic reasoning capabilities of agents with self-play, but they lack analysis of opponent selection in adversarial environments.

Opponent selection has long been recognized as a key factor in adversarial training within the RL community, where methods such as self-play (Silver et al., 2017), opponent sampling (Arulkumaran et al., 2019), and curriculum-based progression (Bengio et al., 2009) are widely used to stabilize learning and encourage generalization. However, these techniques are typically designed for structured, low-dimensional action spaces with well-defined state transitions. In contrast, LLM-based agents operate in an open-ended, high-dimensional language space, where each "action" is a natural language utterance, and each "state" is a rich textual history. This introduces

a qualitatively different set of challenges: opponents not only determine win–lose outcomes, but also shape the semantic distribution and reasoning trajectories the model encounters during learning. First, if opponents are too strong, their strategy distribution may differ too much from the policy's, disrupting learning. Playing with weak opponents provides limited positive feedback, slowing policy updates and trapping the model in suboptimal strategies. Second, weaker opponents can lead to premature convergence to local optima, limiting further improvement. Third, uneven skill levels between agents may also create an imbalanced experience distribution, overemphasizing specific strategies or states while neglecting others. Thus, a well-balanced opponent selection strategy is essential for maintaining a diverse training distribution, providing sufficient learning signals, and ensuring a well-rounded experience, ultimately leading to effective training and robust performance.

To better understand how opponents influence agent learning through interaction in adversarial games, a **play-and-learn** framework is required. First, agents need to interact within an adversarial setting. These interactions generate numerous actions, where actions in a given state reflect the underlying strategy. Second, to refine the strategy, it is crucial to identify which actions are good or bad. Finally, an effective method is needed to update the policy and adjust its action distribution accordingly. Therefore, we propose a new **S**tep-level poli**C**y **O**ptimization method through **P**lay-**A**nd-**L**earn, **SCO-PAL**. In SCO-PAL, we first assign an opponent to the agent and conduct large-scale game interaction, then estimate step-level rewards using Monte Carlo Estimation. Finally, we design a two-stage strategy refinement method using Kahneman-Tversky Optimization (KTO) (Ethayarajh et al., 2024) to optimize the strategy.

Using SCO-PAL, we comprehensively analyze how the opponents affect learning through interaction in strategic environments and find that self-play is the most effective way to conduct the play-and-learn method. In experiments, we select six strategic adversarial games from GTBench to evaluate the strategic reasoning capability of SCO-PAL. By incorporating SCO-PAL with self-play, the average win rate against four downstream opponents increases by about 30% compared to the baselines. Notably, the average win rate against GPT-4 achieves 54.76% across all the games, demonstrating the effectiveness of SCO-PAL in improving the strategic reasoning of language agents.

In conclusion, our contributions are as follows:

- We conduct a comprehensive and detailed analysis of the opponent selection in adversarial environments.
- We design a **S**tep-level poli**C**y **O**ptimization method through **P**lay-**A**nd-**L**earn, **SCO-PAL**, that significantly optimizes the agent's strategy and improves its performance in adversarial games.
- We select multiple strategic games and various opponents to evaluate the competitive performance of SCO-PAL with self-play, finding that the overall win rate increased by about 30% compared to others, reaching 54.76% against GPT-4.

2 Related Work

LLM Agents Large language models (LLMs) are increasingly used to power agents for complex tasks (Zhou et al., 2023; Wang et al., 2023; Zhu et al., 2023; Deng et al., 2024; Putta et al., 2024; Iong et al., 2024; Shen et al., 2024). To improve performance, some methods fine-tune LLMs on expert trajectories (Zeng et al., 2023; Chen et al., 2023), while others adopt a play-and-learn paradigm, allowing agents to learn from interactions (Xi et al., 2024; Song et al., 2024; Xiong et al., 2024; Wang et al., 2024a). These approaches optimize agents via feedback from successful or failed trajectories, using either preference-based or step-wise learning. However, most studies assume static environments; we instead focus on adversarial games, where agents must adapt to dynamic, diverse opponents and reason strategically.

Agent in Adversarial Games Strategic adversarial games require players to employ strategies, pursuing high-level goals that necessitate reasoning and adapting actions based on opponents' behaviors. Huang et al. (2024); Costarelli et al. (2024) collect various strategy games to evaluate agents based on LLMs. They find that agents still fall significantly short in strategic games, not reaching human-level performance. To study the strategic reasoning ability of language agents, Duan et al. (2024) propose GTBench, a language-driven environment designed to evaluate the reasoning capabilities of LLMs in competitive game-theoretic tasks. In this work, we use GTBench as a testbed

and select several games, such as `Liar's Dice` and `Connect4`, for game interaction and evaluation. To imitate the limitation of language agents, existing methods (Wang et al., 2024b; Cheng et al., 2024) train agents by interaction data in gaming environments instead of costly expert-level data. Wang et al. (2024b) proposes to train agents based on the whole successful trajectories, while Cheng et al. (2024) customizes discounted rewards and updates the model through RL. We design a new play-and-learn method SCO-PAL, updating the policy through game interaction, step-level reward estimation, and strategy refinement. Using SCO-PAL with self-play, we achieve the highest average win rates compared to these methods.

3 Preliminary

3.1 Game Modeling

The strategic adversarial games from `GTBench` can be formalized as an episodic Markov Decision Process (MDP), where two players $\mathcal{P} \in \{\mathcal{P}_1, \mathcal{P}_2\}$ take turns performing actions in a gaming environment. Each game is defined by four elements: the state $\mathcal{S}$, representing the current game configuration; the observation $\mathcal{O}$, the visible part of the state for each player; the action space $\mathcal{A}$; and the transition function $T : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$. In state s_i, player $\mathcal{P}_i$ observes o_i and takes an action $a_i \sim \pi_i(\cdot \mid s_i, o_i)$, which transitions the game to state s_{i+1}. The environment returns a reward r when a terminal state is reached. Each player aims to maximize their expected cumulative reward by optimizing their policy π_i. In this work, we focus on improving the policy π_i of the language agent $\mathcal{P}_1$, enabling it to adjust its strategy and achieve higher rewards through interaction.

3.2 Behavioral Cloning

Behavioral cloning (BC) is a supervised learning approach where an agent learns to imitate expert behavior by training on expert-level datasets. Given a dataset $(x, y) \in \mathcal{D}$, the objective of BC is to minimize the following loss:

$$L_{BC}(\pi_\theta; \mathcal{D}) = -\mathbb{E}_{(x,y)\sim\mathcal{D}}[\log \pi_\theta(y|x)], \quad (1)$$

where π_θ is the model to be updated.

3.3 Kahneman-Tversky Optimization (KTO)

Kahneman-Tversky Optimization (KTO) (Ethayarajh et al., 2024) is an offline reinforcement learning method for preference optimization. Unlike Direct Preference Optimization (DPO) (Rafailov et al., 2024), KTO requires only an output and a binary reward indicating desirability for a given input, eliminating the need for pair-wise preference data. This is advantageous in asymmetric scenarios like werewolf (Ye et al., 2025), where creating pair-wise comparisons is challenging. Given the datasets $(x, y) \in \mathcal{D}$, KTO optimizes the policy π_θ using the following loss:

$$r_\theta(x, y) = \log \frac{\pi_\theta(y|x)}{\pi_{ref}(y|x)} \quad (2)$$

$$z_0 = \mathbb{E}_{(x,y)\sim\mathcal{D}}[KL(\pi_\theta(y|x) \| \pi_{ref}(y|x))] \quad (3)$$

$$v(x, y) = \quad (4)$$

$$\begin{cases} \lambda_D \sigma\big(\beta(r_\theta(x, y) - z_0)\big) & \text{if } y \sim y_{\text{desirable}} \mid x \\ \lambda_U \sigma\big(\beta(z_0 - r_\theta(x, y))\big) & \text{if } y \sim y_{\text{undesirable}} \mid x \end{cases} \quad (5)$$

$$L_{KTO}(\pi_\theta; \mathcal{D}) = \mathbb{E}_{(x,y)\sim\mathcal{D}}[\lambda_y - v(x, y)] \quad (6)$$

Here, λ_D and λ_U are hyperparameters representing the losses for desirable and undesirable outcomes, respectively. The parameter λ_y corresponds to λ_D when y is desirable and λ_U when y is undesirable.

4 SCO-PAL

To realize learning from interaction, a proper framework is necessary. In two-player adversarial games, both players engage in strategic reasoning and act to defeat each other, while the environment autonomously determines the game outcome and allocates rewards based on ending conditions. This removes the need for external labeling and facilitates the accumulation of datasets enriched with reward signals. Leveraging this, we design a play-and-learn framework SCO-PAL (Figure 1), using large-scale game interaction to capture strategic optimization directions and improve strategy.

4.1 Game Interaction

The play-and-learn paradigm boosts agents by learning from game interaction with opponents. To realize this, we first conduct interactions between agents in the gaming environments, named Game Interaction (Stage I).

During game interaction, player $\mathcal{P}_i \in \{\mathcal{P}_1, \mathcal{P}_2\}$ performs actions a_t according to policy $\pi_\theta(\cdot \mid s_i, o_i)$ at each time step t. Each action a_t transitions the game from state s_t to s_{t+1}, and the environment concludes the game by determining the ending condition and providing a reward r, indicating a win, loss, or tie. For player $\mathcal{P}_i$, a trajectory τ is denoted

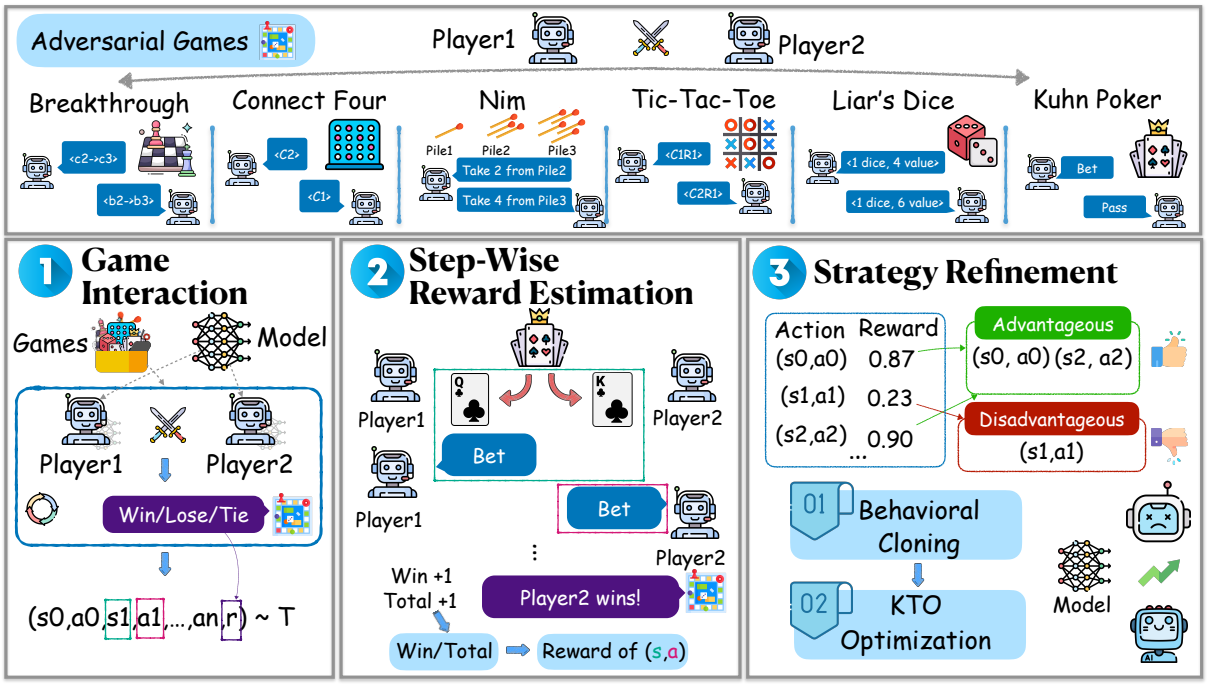


Figure 1: Overview of our play-and-learn method, SCO-PAL. SCO-PAL enables language agents to improve through game interaction via a three-stage process: Stage I Game Interaction assigns an opponent and conducts large-scale gameplay; Stage II Step-Wise Reward Estimation computes step-level rewards; Stage III Strategy Refinement updates the policy via behavioral cloning and KTO Optimization.

as $(s_0, a_1, s_1, a_2, \ldots, a_n, s_n, r)$, where r signifies the outcome reward. Through interaction, we collect a set of trajectories $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_n\}$ for strategy refinement.

4.2 Step-Wise Reward Estimation

The environment provides only outcome-level rewards, leaving intermediate actions unlabeled. To address this, we design a step-level reward estimation method in Step-Wise Reward Estimation (Stage II) and compare different estimation strategies in § 7.2.

Step-wise reward estimation identifies whether an action is advantageous or disadvantageous, guiding subsequent strategy refinement. We apply Monte Carlo Estimation over interaction trajectories $\mathcal{T}$, estimating the reward $r(s_i, a_i)$ of each $(s_i, a_i) \in \tau$ by its empirical win rate. Specifically, we count total occurrences N_{all} and winning occurrences N_{win} of (s_i, a_i) in $\mathcal{T}$, and compute: $r(s_i, a_i) = \frac{N_{win}}{N_{all}}$.

4.3 Strategy Refinement

After reward estimation, we obtain trajectories with step-level rewards for training. In Strategy Refinement (Stage III), we propose a two-stage strategy improvement method: (1) **Behavioral Cloning** adapts the agent to the game environment using state-action pairs (s_i, a_i) with rewards $r(s_i, a_i)$ above a predefined threshold δ as advantageous actions for training. (2) **KTO Optimization** formulates learning as preference optimization. We label state-action pairs as advantageous (desirable) or disadvantageous (undesirable) based on $r(s_i, a_i)$ and δ, encouraging the agent to prefer advantageous actions while avoiding disadvantageous ones.

In experiments, we find that Behavioral Cloning enables fast adaptation, while KTO Optimization helps the agent effectively distinguish between action qualities, significantly improving win rates in downstream competitions.

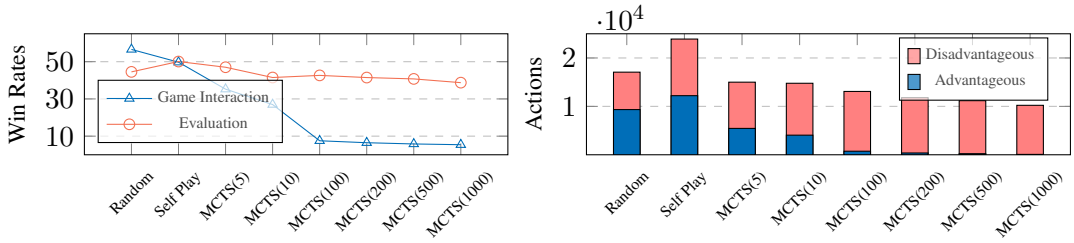
5 Analysis on Opponent Selection

5.1 Setup

Game Interaction Opponents To study how opponent strength affects learning, we introduce level-controllable symbolic opponents ranging from random (weak) to MCTS-based (strong). The strength of MCTS is adjusted by varying the maximum number of simulations (`max_simulation`)[1], set to 5, 10, 100, 200, 500, and 1000.

Game Benchmark We select 6 games from `GTBench` (Duan et al., 2024): (1) `Breakthrough`, (2) `Connect Four`, (3) `Tic-Tac-Toe`, (4) `Kuhn`

[1]Details in Appendix B.1.



(a) Win rates in game interaction and evaluation. (b) Number of actions collected from different opponents.

Figure 2: Analysis of opponent selection in game interaction. (a) *Game Interaction* shows the base model's win rates against different opponents, while *Evaluation* shows the win rates of the model trained with SCO-PAL against four downstream opponents. (b) displays the number of actions collected when interacting with different opponents. The number in MCTS(·) indicates the `max_simulation` setting.

`Poker`, (5) `Liar's Dice`, and (6) `Nim`[2]. Each game interaction involves 1,000 episodes, and each evaluation is conducted over 100 episodes per opponent. To mitigate first-player advantage, players alternate roles and play an equal number of games.

Model Setting We use Qwen2-7B-Instruct (qwe, 2024) as the base model. During game interaction, the temperature is set to 0.7 (see § A.4 for ablation). For evaluation, we follow `GTBench` (Duan et al., 2024) and set the temperature to 0.2. We also demonstrate the effectiveness of SCO-PAL on other models (*e.g.*, Mistral) in the Appendix A.8.

Parameters The threshold for identifying advantageous actions is 0.5. In Behavioral Cloning, we set the learning rate to 1e-6, batch size to 2, warm-up ratio to 0.1, and train for 5 epochs. For KTO, the batch size is 2, the gradient accumulation steps are 8, the learning rate is 1e-6, and training also lasts 5 epochs. The loss weights λ_D and λ_U are set such that $\lambda_D n_D = \lambda_U n_U$, where n_D and n_U are the numbers of advantageous and disadvantageous samples; the larger weight is set to 1.0, and the smaller scaled proportionally. All experiments are conducted on 8 NVIDIA A100 GPUs (80GB each).

Metrics We evaluate performance using win rate. For each game, we record outcome counts N_{win}, N_{lose}, and N_{tie}, and compute the win rate as: $w = \frac{N_{win}+0.5N_{tie}}{N_{win}+N_{lose}+N_{tie}}$.

Evaluation Opponents We evaluate against four opponents: (1) *Random*, selecting actions at random; (2) *MCTS*, configured with 1 rollout, UCT parameter $c = 2$, and max simulations = 1000; (3) *GPT-3.5* (GPT-3.5-turbo-0125) (OpenAI et al., 2024), an OpenAI language model; (4) *GPT-4* (GPT-4-turbo-2024-04-09) (OpenAI et al., 2024), a more advanced OpenAI model.

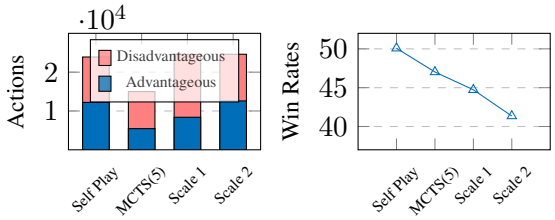
[2]Game introductions are in Appendix C.1.

5.2 Results

Opponents affect the number and proportion of actions. We first examine win rates against different opponents. As shown in Figure 2(a), the language agent achieves a 56.59% win rate against the weak opponent (random) and nearly 50% in self-play. Against strong opponents (MCTS), as `max_simulation` increases, MCTS yields more accurate state estimations, and the agent's win rate drops from 35.24% to 5.40%, consistent with the expected strength levels.

We then analyze the number of actions collected during the interaction. As shown in Figure 2(b), the number of actions decreases when facing either very weak or very strong opponents. In both cases, the skill gap makes outcomes more predictable, with the advantaged player repeatedly winning through fixed strategies, thereby reducing the diversity of actions. In contrast, self-play yields the most balanced ratio of advantageous to disadvantageous actions. This ratio becomes increasingly skewed against stronger opponents, where the agent struggles to generate advantageous actions, resulting in a significant imbalance.

Self-play is the most effective choice. We evaluate SCO-PAL by training it with different opponents and reporting the average win rate against four downstream opponents. As shown in Figure 2(a), the results mirror the action count trends in Figure 2(b), with self-play yielding the largest performance improvement, outperforming both weak and strong opponents. As opponent strength increases, the diversity and proportion of advantageous actions decline, leading to reduced learning



(a) Number of actions. (b) Win rates after training.

Figure 3: Evaluation of different scaling strategies for balancing advantageous and disadvantageous samples. (a) shows the number of actions obtained under each setting. (b) reports the win rates against four downstream opponents using agents trained on the corresponding data.

effectiveness. These findings underscore the importance of collecting diverse gameplay data and maintaining a balanced distribution of advantageous and disadvantageous actions for effective strategy optimization.

Scaling data does not improve performance. Figure 2(b) shows that self-play yields the richest and most balanced data, leading to more accurate strategy optimization. To examine the effect of data volume, we upsample [3] training data under different opponent settings, as shown in Figure 3(a). Specifically, we consider two scaling strategies: (1) **Scale 1** aligns the total number of samples with that of self-play while preserving the original advantageous-to-disadvantageous ratio. (2) **Scale 2** directly matches the number of advantageous and disadvantageous samples to those from self-play, thereby eliminating action-ratio differences induced by opponent strength. In both settings, we use actions collected from MCTS(5) interactions in Figure 2(b) as the base data.

We train agents on the scaled data and compare performance with self-play and unscaled MCTS(5) in Figure 2(a), with results summarized in Figure 3(b). We observe that Scale 1 leads to a noticeable performance drop, suggesting that simply increasing sample size without improving action quality can cause overfitting and misguide optimization. Likewise, Scale 2 results in a further 3% decline in win rate, indicating that forcibly balancing the number of samples disrupts the original optimization trajectory. These findings suggest that data scaling alone cannot compensate for the lack of high-quality or diverse actions—balance and strategic richness must be intrinsically present in the data.

Potential of combining strong-opponent imitation and SCO-PAL. In our opponent selection analysis, strong opponents play a key role. To further explore this, we examine the combination of SCO-PAL with imitation learning from strong opponents. Results in Appendix A.1 and A.2 show that imitation can complement SCO-PAL, offering further performance improvements.

6 Experiments on SCO-PAL with Self-Play

Analysis in § 5 demonstrates that self-play is the most effective way to conduct play-and-learn in adversarial games. To fully explore the performance of SCO-PAL with self-play, we compare SCO-PAL under self-play with existing play-and-learn methods, both against downstream opponents, and conduct a head-to-head evaluation in this section.

6.1 Setup

Opponents In game interaction, we set the opponent as the language agent itself (self-play setting in § 5). Other settings are the same as these in § 5.

Baselines We select three baselines: (1) Base model: we use Qwen2-7b-Chat as the base model; (2) Behavioral Cloning (BC): fine-tuning the model on successful trajectories from game interactions inspired by Wang et al. (2024b); (3) SPAG (Cheng et al., 2024): that updates the policy by hand-crafted rewards with a discounted coefficient [4].

6.2 Main Results

In Table 1, the base model's average win rate against the four downstream opponents is generally lower, only slightly beating random with a rate of 52.44%. After training, SCO-PAL achieves the most significant improvement, 50.08%, surpassing 43.50% of BC and 41.70% of SPAG, and defeating GPT-4 with an average winning rate of 54.76%. The reason behind the improvement is that both BC and SPAG struggle with learning actions due to inaccurate reward estimation and insufficient optimization methods. For example, in BC, successful trajectories might include suboptimal actions. In SPAG, manually specified coefficients can become inaccurate and not general due to changes in environments. SCO-PAL uses more general reward

[3] For a given dataset D1, we randomly sample data from D1 to construct a new dataset D2 of the expected upsampling size.

[4] Detailed introduction of baselines are in Appendix B.2.

Methods	Opponents	Breakthrough	Connect Four	Tic-Tac-Toe	Kuhn Poker	Liar's Dice	Nim	Avg.
Base	GPT-4	7.22	46.46	44.50	20.90	20.31	31.00	28.40
	GPT-3.5	57.65	46.32	55.26	18.75	21.54	53.00	42.09
	Random	66.67	61.62	63.64	39.06	39.66	44.00	52.44
	MCTS	2.74	0.00	6.00	20.90	4.84	0.00	5.75
	AVG	33.57	38.60	42.35	24.90	21.59	32.00	32.17
BC	GPT-4	26.00	32.50	64.00	45.00	57.00	31.00	42.58
	GPT-3.5	52.63	10.10	100.00	45.00	68.69	44.68	53.52
	Random	68.09	66.00	78.00	61.00	83.00	50.00	67.68
	MCTS	3.30	0.00	0.00	45.00	13.00	0.00	10.22
	AVG	37.50	27.15	**60.50**	**49.00**	55.42	31.42	43.50
SPAG	GPT-4	29.87	42.00	33.00	41.00	57.00	39.00	40.31
	GPT-3.5	30.14	29.29	98.00	37.00	64.65	44.00	50.51
	Random	71.43	65.00	75.00	45.45	82.00	57.00	65.98
	MCTS	0.00	0.00	0.00	45.00	15.00	0.00	10.00
	AVG	32.86	34.07	51.50	42.11	54.66	35.00	41.70
SCO-PAL	GPT-4	52.53	62.63	71.43	45.00	60.00	37.00	54.76
	GPT-3.5	60.42	72.00	54.64	45.00	80.00	77.00	64.84
	Random	85.88	76.00	76.02	35.00	89.00	59.00	70.15
	MCTS	2.33	0.00	10.10	45.00	6.00	0.00	10.57
	AVG	**50.29**	**52.66**	53.05	42.50	**58.75**	**43.25**	**50.08**

Table 1: Win rates against four opponents on six games. The best results are **bolded**, and the second best ones are underlined. Statistical significance tests are reported in the Appendix A.6.

	Base	BC	SPAG	SCO-PAL	Avg.
Base	50.00	27.20	24.21	22.97	31.10
BC	72.80	50.00	51.03	45.29	54.78
SPAG	75.79	48.97	50.00	47.31	55.52
SCO-PAL	**77.03**	**54.71**	**52.69**	50.00	**58.61**

Table 2: The vertical axis represents Player 1, the horizontal axis represents Player 2, and the values indicate the win rate of Player 1 against Player 2.

estimates and finer-grained reward signals, using KTO optimization, leading to more comprehensive learning. We also calculate regret, with results in the Appendix A.9 showing that SCO-PAL reduces regret and improves strategic reasoning.

6.3 Head-to-Head Evaluation

Settings In this experiment, we use the six games from `GTBench` to conduct direct battles between two players, with 100 games played in each match.

Results As shown in Table 2, SCO-PAL achieves a win rate of over 50% compared to all the others. While BC and SPAG can also outperform the base model, both of them struggle against our SCO-PAL. This indicates that our approach effectively identifies the advantageous and disadvantageous actions and uses step-level strategy optimization to achieve a consistently high win rate.

6.4 Generalization on Unseen Games

Settings In § 6.2, SCO-PAL demonstrates effectiveness on in-domain games, showing its improvement in the language agent's strategic reasoning ability. To explore the generalization potential of SCO-PAL, we further select three unseen games, `Blind Auction`, `Pig` and `Prisoner's Dilemma` [5] from `GTbench` and test the performance of SCO-PAL against a random strategy.

Results As shown in Table 3, the strategic reasoning ability developed by SCO-PAL also transfers to unseen games, with an average improvement of 8% across three games. In these games, we find that the agent can make reasonable decisions and timely strategy adjustments based on the current state and winning goals, even being unseen before, indicating that SCO-PAL enhances the language agent's strategic reasoning capability [6].

6.5 Iteration of SCO-PAL with Self-Play

We analyze the performance of applying SCO-PAL with self-play iteratively and find that as iterations increase, the agent's performance initially improves but later declines. This demonstrates that our SCO-PAL supports effective iteration but carries a slight risk of overfitting due to RL training. See the Appendix A.3 for detailed experiments.

[5] Introductions of unseen games are in appendix C.2.

[6] Experiments on general capabilities are in Appendix A.7.

	Auc.	Pig	Pri.	Avg.
Base	**71.00**	88.00	37.00	65.00
SCO-PAL	67.00	**92.00**	**60.00**	**73.00**

Table 3: Evaluation on unseen games. **Auc.** refers to `Blind Auction`; **Pig** refers to `Pig`; **Pri.** refers to `Prisoner's Dilemma`.

7 Ablations

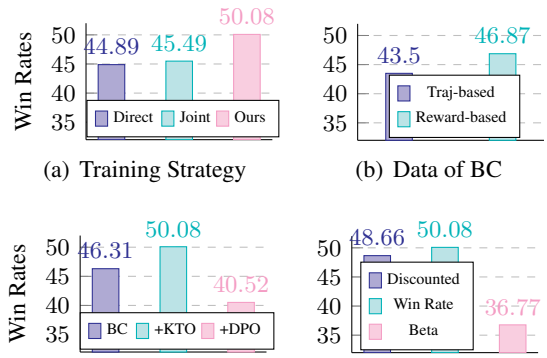
7.1 Ablation of Training Strategies in SCO-PAL

To validate the effectiveness of our two-stage approach, we conduct ablations on both the training procedure and the data used for BC.

Ablation of Training Strategy We compare three training strategies: (1) **Direct KTO**, applying KTO without BC; (2) **Joint Loss**, combining BC and KTO objectives; (3) **Two-Stage (Ours)**, performing BC followed by KTO. As shown in Table 4(a), adding BC improves adaptation to the game environment, with the two-stage strategy achieving a 4.59% performance gain over the joint-loss variant. These results suggest that initializing with BC provides a more stable foundation for subsequent preference optimization.

Ablation of Data for Behavioral Cloning In SCO-PAL, BC is trained on advantageous actions identified from game interaction. We compare two BC data construction methods based on self-play: (1) **Trajectory-Based**, which uses all actions from successful trajectories; (2) **Reward-Based**, which filters actions with estimated rewards exceeding a threshold. As shown in Table 4(b), using only advantageous actions yields better results, as successful trajectories may still contain suboptimal decisions. This highlights the importance of fine-grained filtering in BC data selection.

Ablation of Strategy Optimization We compare strategy optimization methods within SCO-PAL and find that adding KTO on top of BC significantly improves performance by reinforcing advantageous actions and suppressing disadvantageous ones, as shown in Table 4(c). This highlights the importance of strategy optimization after self-play. We further compare KTO with DPO and observe a 9.56% gain in average win rate. Unlike DPO, which relies on pairwise comparisons under identical states, KTO operates at the step level and offers broader coverage of action quality, leading to more effective optimization.



(a) Training Strategy (b) Data of BC

(c) Preference Optimization (d) Reward Estimation

Figure 4: Ablations of SCO-PAL.

7.2 Ablation of Reward Estimation Methods

Different reward estimation methods produce varying distributions of advantageous and disadvantageous actions, thus influencing SCO-PAL's performance. We compare three approaches—**Discounted Reward**, **Win Rate**, and **Beta**—as detailed in Appendix B.3. As shown in Figure 4(d), the win rate method performs best with a 50.08% win rate, compared to 48.66% for discounted reward and 36.77% for beta. Win rate directly captures success likelihood, aligning well with the game objective and offering a stable, interpretable signal. In contrast, discounted reward requires careful tuning of the discount factor, while the beta method depends on sufficient data and sensitive hyperparameters, reducing practicality. Overall, win rate offers the best trade-off between simplicity, stability, and alignment, making it the most effective choice in SCO-PAL.

8 Conclusion

In this paper, we focus on improving the strategic reasoning ability of language agents under the play-and-learn paradigm. We design a **S**tep-level poli**C**y **O**ptimization method through **P**lay-**A**nd-**L**earn, SCO-PAL. Using SCO-PAL, we analyze how opponents influence agent learning in adversarial games and find that self-play yields the best results. Incorporating SCO-PAL with self-play, we boost the average win rate by 30% compared to baselines and achieve a 54.76% win rate against GPT-4. Our work provides insights into opponent selection in adversarial environments and introduces a new learning framework, offering an effective way to enhance the strategic reasoning of language agents.

Limitations

While SCO-PAL demonstrates strong empirical performance in symbolic adversarial games, it has certain limitations. First, our study is limited to well-defined, turn-based games and does not explore more open-ended or partially observable environments such as negotiation or web-based tasks. Second, the opponent pool is restricted to scripted agents and LLM variants; incorporating human or style-diverse LLM opponents could further enrich strategic diversity but introduces additional challenges. We leave these directions for future work.

Ethical Statement

This work focuses on improving the strategic reasoning abilities of language agents in adversarial games using self-play and interaction-based learning. All experiments are conducted in controlled, simulated environments without real user data or deployment. Our use of existing artifacts is consistent with their intended use, and we follow their license and terms. No sensitive data or human annotations are used. While advances in strategic language modeling may enable more capable agents in competitive settings, we caution against misuse in domains such as manipulation, deception, or adversarial dialogue generation. We encourage future research to incorporate safety constraints and ethical safeguards when deploying such agents in open environments.

References

2024. Qwen2 technical report.

Kai Arulkumaran, Antoine Cully, and Julian Togelius. 2019. Alphastar: an evolutionary computation perspective. *Proceedings of the Genetic and Evolutionary Computation Conference Companion*.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. In *International Conference on Machine Learning*.

Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck. 2008. Monte-carlo tree search: A new framework for game ai. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 4, pages 216–217.

Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. 2023. Fireact: Toward language agent fine-tuning. *arXiv preprint arXiv:2310.05915*.

Pengyu Cheng, Tianhao Hu, Han Xu, Zhisong Zhang, Yong Dai, Lei Han, and Nan Du. 2024. Self-playing adversarial language game enhances llm reasoning. *arXiv preprint arXiv:2404.10642*.

Anthony Costarelli, Mat Allen, Roman Hauksson, Grace Sodunke, Suhas Hariharan, Carlson Cheng, Wenjie Li, Joshua Clymer, and Arjun Yadav. 2024. Gamebench: Evaluating strategic reasoning abilities of llm agents. *arXiv preprint arXiv:2406.06613*.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. 2024. Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems*, 36.

Jinhao Duan, Renming Zhang, James Diffenderfer, Bhavya Kailkhura, Lichao Sun, Elias Stengel-Eskin, Mohit Bansal, Tianlong Chen, and Kaidi Xu. 2024. Gtbench: Uncovering the strategic reasoning limitations of llms via game-theoretic evaluations. *arXiv preprint arXiv:2402.12348*.

Kawin Ethayarajh, Winnie Xu, Niklas Muennighoff, Dan Jurafsky, and Douwe Kiela. 2024. Kto: Model alignment as prospect theoretic optimization. *arXiv preprint arXiv:2402.01306*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. *Preprint*, arXiv:2009.03300.

Jen-tse Huang, Eric John Li, Man Ho Lam, Tian Liang, Wenxuan Wang, Youliang Yuan, Wenxiang Jiao, Xing Wang, Zhaopeng Tu, and Michael R Lyu. 2024. How far are we on the decision-making of llms? evaluating llms' gaming ability in multi-agent environments. *arXiv preprint arXiv:2403.11807*.

Iat Long Iong, Xiao Liu, Yuxuan Chen, Hanyu Lai, Shuntian Yao, Pengbo Shen, Hao Yu, Yuxiao Dong, and Jie Tang. 2024. OpenWebAgent: An open toolkit to enable web agents on large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pages 72–81, Bangkok, Thailand. Association for Computational Linguistics.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b. *Preprint*, arXiv:2310.06825.

Bill Yuchen Lin, Yicheng Fu, Karina Yang, Faeze Brahman, Shiyu Huang, Chandra Bhagavatula, Prithviraj Ammanabrolu, Yejin Choi, and Xiang Ren. 2024. Swiftsage: A generative agent with fast and slow thinking for complex interactive tasks. *Advances in Neural Information Processing Systems*, 36.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, and 262 others. 2024. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

Pranav Putta, Edmund Mills, Naman Garg, Sumeet Motwani, Chelsea Finn, Divyansh Garg, and Rafael Rafailov. 2024. Agent q: Advanced reasoning and learning for autonomous ai agents. *arXiv preprint arXiv:2408.07199*.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, and 1 others. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2024. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2024. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems*, 36.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. 2017. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *Preprint*, arXiv:1712.01815.

Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. 2024. Trial and error: Exploration-based trajectory optimization for llm agents. *arXiv preprint arXiv:2403.02502*.

Chaojie Wang, Yanchen Deng, Zhiyi Lyu, Liang Zeng, Jujie He, Shuicheng Yan, and Bo An. 2024a. Q*: Improving multi-step reasoning for llms with deliberative planning. *arXiv preprint arXiv:2406.14283*.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*.

Ruiyi Wang, Haofei Yu, Wenxin Zhang, Zhengyang Qi, Maarten Sap, Graham Neubig, Yonatan Bisk, and Hao Zhu. 2024b. Sotopia-π: Interactive learning of socially intelligent language agents. *arXiv preprint arXiv:2403.08715*.

Zhiheng Xi, Yiwen Ding, Wenxiang Chen, Boyang Hong, Honglin Guo, Junzhe Wang, Dingwen Yang, Chenyang Liao, Xin Guo, Wei He, and 1 others. 2024. Agentgym: Evolving large language model-based agents across diverse environments. *arXiv preprint arXiv:2406.04151*.

Weimin Xiong, Yifan Song, Xiutian Zhao, Wenhao Wu, Xun Wang, Ke Wang, Cheng Li, Wei Peng, and Sujian Li. 2024. Watch every step! llm agent learning via iterative step-level process refinement. *arXiv preprint arXiv:2406.11176*.

Rong Ye, Yongxin Zhang, Yikai Zhang, Haoyu Kuang, Zhongyu Wei, and Peng Sun. 2025. Multi-agent kto: Reinforcing strategic interactions of large language model in language game. *Preprint*, arXiv:2501.14225.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. 2023. Agenttuning: Enabling generalized agent abilities for llms. *arXiv preprint arXiv:2310.12823*.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, and 1 others. 2023. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*.

Xizhou Zhu, Yuntao Chen, Hao Tian, Chenxin Tao, Weijie Su, Chenyu Yang, Gao Huang, Bin Li, Lewei Lu, Xiaogang Wang, and 1 others. 2023. Ghost in the minecraft: Generally capable agents for open-world environments via large language models with text-based knowledge and memory. *arXiv preprint arXiv:2305.17144*.

Appendix

A Supplementary Experiments

A.1 Imitating Strong Players

In § 5, we examine how opponents influence performance by introducing stronger ones. Strong opponents tend to adopt better strategies, leading to higher win rates. To further leverage their potential, we explore the effect of imitating strong players.

We first experiment with direct imitation. As shown in § 5, self-play generates diverse, balanced, and high-quality data through interactive gameplay. To better capture strong players' strategies, we adapt SCO-PAL accordingly. Two strong agents of equal strength play against each other, followed by step-wise reward estimation. We then select advantageous samples with win rates above a threshold for BC training. As shown in Figure 5, learning from strong players significantly improves win rates.

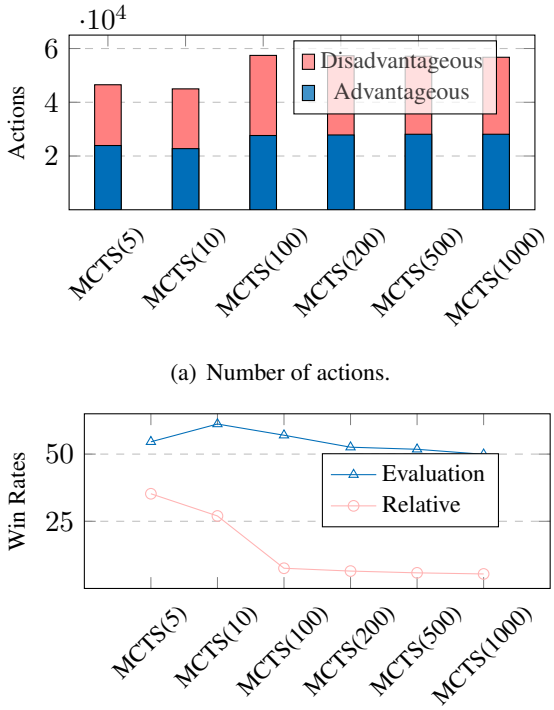
Moreover, we observe that as the opponent's relative strength (compared to the base model) increases, the agent's performance initially improves but later declines. This indicates that while imitating strong players is beneficial, a large skill gap and divergent strategies can impede effective learning.

A.2 Combining SCO-PAL and Imitating Strong Players

In §A.1, we observed that learning from strong players enhances strategic reasoning. Here, we take the best-performing setting—imitating MCTS(5)—and apply SCO-PAL with self-play. We evaluate the resulting model against a strong opponent, MCTS(1000). As shown in Figure 6, SCO-PAL remains effective even after expert imitation, improving the average win rate against strong opponents by 2.44%. This suggests that imitating strong players can complement SCO-PAL with self-play, leading to further gains.

A.3 Discussion on Iteration

Settings During iteration, the agent plays against historical models in a play-and-learn setup. In the first round, as there is no historical model, the base model (Qwen2-7B-Chat) performs self-play, and we apply SCO-PAL to obtain *Iter1*. In the second round, *Iter1* plays against the base model, is trained to yield *Iter2*, and then *Iter2* plays against *Iter1* to produce *Iter3*.



(a) Number of actions.

(b) Win rates: relative and evaluation.

Figure 5: (a) shows the number of actions taken during self-play between strong players of varying levels. In (b), *Relative* denotes the win rate of the base model against different opponents, while *Evaluation* indicates the average win rate of the model trained by imitating strong players, evaluated against four downstream opponents. The number in MCTS(·) specifies the `max_simulation` parameter.

Results Figure 7 presents the results of iteratively applying SCO-PAL. We observe that iterative self-play with SCO-PAL gradually improves the model's strategic performance. However, in *Iter3*, the win rate declines slightly. We attribute this to overfitting introduced by repeated RL training (Zeng et al., 2023; Xiong et al., 2024), which may narrow the model's strategic diversity and reduce effectiveness.

A.4 Impact of Temperature

Temperature is a key parameter in LLM generation, influencing the diversity of outputs. To assess its effect on SCO-PAL, we evaluate five settings: 0.2, 0.5, 0.7, 1.0, and 1.2, and systematically analyze how temperature impacts performance.

Temperature affects action diversity and game validity. Temperature controls the token sampling behavior of language models. Lower temperatures lead to more deterministic outputs, while higher values introduce greater diversity. In our set-

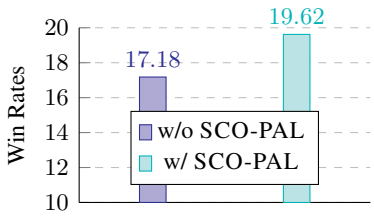


Figure 6: Win rate changes against MCTS(1000) after applying SCO-PAL with self-play to the model trained by imitating MCTS(5).

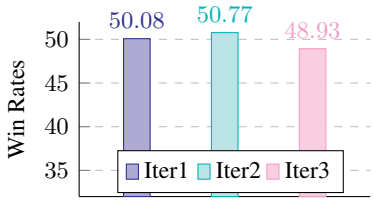


Figure 7: Performance across iterations of SCO-PAL. Each iteration involves self-play and training against previous models.

ting, temperature influences the diversity of actions during game interaction by the language agent, which in turn affects the number of actions taken.

As shown in Figure 9(a), action diversity increases with temperature up to a point, peaking between 0.7 and 1.0, then declines. To understand this trend, we examine the rate of valid games—those where the agent's actions are successfully parsed by the environment. At high temperatures, the agent is more likely to generate invalid actions, reducing the number of valid interactions. Figure 9(b) shows that the valid rate peaks between 0.5 and 0.7.

Based on this analysis, we set the temperature to 0.7 during self-play. We also evaluate SCO-PAL under different temperature settings, as shown in Figure 9(c). Results confirm that a temperature of 0.7 yields the highest post-training win rate, indicating that a balance between stability and diversity of valid actions is crucial for optimal performance.

A.5 Impact of Game Balance

Different games have varying action spaces, leading to an unequal number of actions after self-play.

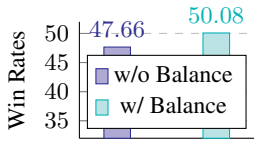


Figure 8: Comparison of win rates with and without the balance mechanism.

	P-Value
BC	0.0017
SPAG	0.0110

Table 4: Significance test for the main experiment.

	Accuracy.
Base	70.54
SCO-PAL	70.65

Table 5: Model performance on a general benchmark (MMLU).

To avoid biased learning, we apply data balancing by upsampling actions from games with fewer actions and downsampling those with more, while keeping the total sample size constant. We compare results before and after balancing. As shown in Figure 8, this strategy ensures the language agent learns effectively across different games.

A.6 Statistical Significance Tests

We perform significance tests comparing our method against baselines (BC and SPAG). As shown in Table 4, all p-values are below 0.05, indicating that the improvements are statistically significant.

A.7 Generalization Capability Evaluation

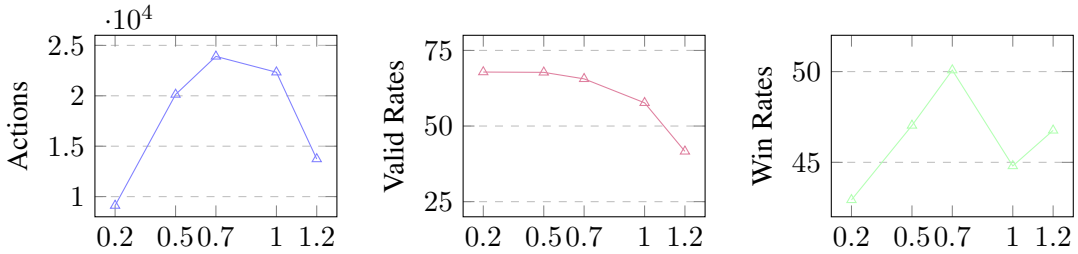
We assess whether applying SCO-PAL affects the model's generalization. Specifically, we evaluate performance on MMLU (Hendrycks et al., 2021) before and after training. As shown in Table 5, SCO-PAL yields a slight improvement over the base model. This suggests some generalization to unseen tasks, although the gain remains limited due to the breadth and diversity of the benchmark.

A.8 SCO-PAL on Other Models

To test the transferability of SCO-PAL, we apply it to Mistral-7B-Instruct-v0.3 (Jiang et al., 2023). As shown in Table 6, SCO-PAL also improves strategic reasoning in Mistral, demonstrating its versatility across different base models.

	Average Win Rates.
Base	32.18
SCO-PAL	45.80

Table 6: SCO-PAL on Mistral.



(a) Number of actions (b) Game validity rate (c) Post-training win rate

Figure 9: Effect of temperature on (a) number of actions during interaction, (b) validity rate of generated games, and (c) average win rate after training with data from each temperature setting.

	Nim	Tic-Tac-Toe
Base	0.0564	0.0658
SCO-PAL	**0.0378**	**0.0472**

Table 7: Regret on Nim and Tic-Tac-Toe.

	Average Win Rates
Base	19.54
SCO-PAL	**37.62**

Table 8: Win Rates against DeepSeek-R1 (Guo et al., 2025).

A.9 Regret Analysis

To assess progress toward equilibrium, we measure regret in two benchmark games—Nim and Tic-Tac-Toe—before and after applying SCO-PAL. Results in Table 7 show consistent regret reduction, indicating enhanced strategic alignment.

A.10 Comparison with Reasoning Models

To explore whether our method effectively improves strategic reasoning capabilities, we evaluate the win rates against strong reasoning models with enhanced planning and decision-making capabilities. As shown in Table 8, although it is always at a disadvantage, SCO-PAL can still learn advantageous actions and improve itself through training.

B Method Settings

B.1 MCTS Search Algorithm

Monte Carlo Tree Search (MCTS) (Chaslot et al., 2008) is a heuristic search algorithm used for decision-making processes, particularly in game playing. The core principle of MCTS is to build a search tree incrementally and asymmetrically by using random simulations to explore the most promising moves. The algorithm consists of four main steps: selection, expansion, simulation, and backpropagation.

- **Selection:** Starting from the root node, select child nodes based on a policy that balances exploration and exploitation. The most common policy is the Upper Confidence Bound for Trees (UCT), defined as:

$$UCT = \frac{w_i}{n_i} + c \times \sqrt{\frac{\ln N}{n_i}}$$

  where w_i is the number of wins for the node, n_i is the number of times the node has been visited, N is the total number of simulations, and c is the exploration constant.

- **Expansion:** Once a leaf node is reached, expand the tree by adding one or more child nodes corresponding to possible moves. In our setup, the tree expands with each new node representing possible future states.

- **Simulation:** From the new node, perform a rollout, i.e., play out the game randomly until a terminal state is reached. The outcome of this simulation provides an estimate of the node's value.

- **Backpropagation:** Use the result of the simulation to update the information in the nodes along the path from the expanded node back to the root. This involves updating win ratios and visit counts to reflect the newly gathered information.

In our settings, we set `rollout_count`: The number of rollouts performed from each new node is set to 1; `c`: The exploration constant of UCT is set to 2, balancing the trade-off between exploration of new nodes and exploitation of known promising nodes; `max_simulations`: The maximum number of simulations. This parameter controls the total number of simulations that can be performed during decision-making processes. By adjusting `max_simulations`, we can control the level of agents, where a higher number of simulations generally leads to more informed and strategic decisions, enhancing the agent's competitive performance.

B.2 Baselines

BC Inspired by (Wang et al., 2024b), BC involves using the action sequences from winning players as training data to improve the agent's strategy. This method can be broken down into the following steps:

First, we collect a dataset $\mathcal{D}$ comprising trajectories $\mathcal{T}$ from games where the player wins. Each trajectory τ is represented as a sequence of state-action pairs:

$$(s_0, a_1, s_1, a_2, s_2, \ldots, a_n, s_n, r)$$

where r is 1, representing that the agent wins.

The goal is to train the policy π_θ to replicate these successful behaviors. The learning objective is to minimize the Behavioral Cloning loss, which is defined as:

$$L_{BC} = -\mathbb{E}_{(s,a)\sim\mathcal{D}}[\log \pi_\theta(a|s)]$$

Where π_θ is the policy model that we aim to update.

SPAG (Cheng et al., 2024) The given formula represents the loss function $\mathcal{L}_{\text{SPAG}}(\pi_\theta)$, which is used to optimize the policy π_θ in a strategic game setting. The loss function is composed of several components:

$$\mathcal{L}_{\text{SPAG}}(\pi_\theta) =$$

$$-\frac{1}{2}\mathbb{E}_{\mathcal{T}^1_{\bar{\theta}}}\left[\sum_{t=1}^{T}\frac{\pi_\theta(\mathbf{u}_t|f_1(\mathcal{S}_{t-1}))}{\pi_{\bar{\theta}}(\mathbf{u}_t|f_1(\mathcal{S}_{t-1}))}\hat{A}_t^{\mu_{\bar{\theta}}} - \beta_2 KL[\pi_\theta\|\pi_{\bar{\theta}}]\right]$$

$$-\frac{1}{2}\mathbb{E}_{\mathcal{T}^2_{\bar{\theta}}}\left[\sum_{t=1}^{T}\frac{\pi_\theta(\mathbf{v}_t|f_2(\mathcal{S}'_t))}{\pi_{\bar{\theta}}(\mathbf{v}_t|f_2(\mathcal{S}'_t))}\hat{A}_t^{\nu_{\bar{\theta}}} - \beta_2 KL[\pi_\theta\|\pi_{\bar{\theta}}]\right]$$

where:$\mathbb{E}_{\mathcal{T}^1_{\bar{\theta}}}$ and $\mathbb{E}_{\mathcal{T}^2_{\bar{\theta}}}$ denote the expectations over trajectories for the player1 and player2, respectively. π_θ is the policy being optimized, and $\pi_{\bar{\theta}}$ is a reference policy. $\mathbf{u}_t$ and $\mathbf{v}_t$ are actions taken by player1 and player2 at time t. f_1 and f_2 are the prompts for player1 and player2. $\hat{A}_t^{\mu_{\bar{\theta}}}$ and $\hat{A}_t^{\nu_{\bar{\theta}}}$ are approximations of the advantage functions for player1 and player2. β_2 is a regularization parameter for the Kullback-Leibler divergence $KL[\pi_\theta\|\pi_{\bar{\theta}}]$. In SPAG, they define $\hat{A}_t^{\mu_{\bar{\theta}}} \approx r(s_{t-1}, u_t)$ and $\hat{A}_t^{\nu_{\bar{\theta}}} \approx -r(s'_t, v_t)$.

Given a complete trajectory, the rewards are assigned for each action:

$$r(s_{t-1}, u_t) = \begin{cases} (1-\gamma)\gamma^{T-t}/(1-\gamma^{T+1}), & \text{if player1 wins,} \\ -(1-\gamma)\gamma^{T-t}/(1-\gamma^{T+1}), & \text{if player2 wins,} \\ 0, & \text{if game is tied.} \end{cases}$$

$$r(s'_t, v_t) = \begin{cases} -r(s_{t-1}, u_t), & \text{if attacker wins,} \\ -r(s_{t-1}, u_t), & \text{if defender wins,} \\ 0, & \text{if game is tied.} \end{cases}$$

In our experiments, we set γ to 0.8, the same as the setting in SPAG.

B.3 Reward Estimation Methods

Discounted Reward Discounted reward is used to evaluate the long-term value of actions. The discounted reward R_t from time step t is the sum of all future rewards, each multiplied by a discount factor γ:

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{T-t} r_T$$

If rewards are given only at the end of the game, then $r_k = 0$ for $k < T$, with a final reward r_T based on the outcome. For a state-action pair (s, a) occurring at time t, the discounted reward is:

$$R(s, a) = \gamma^{(T-t)} \times R_T$$

Calculation Formula:

$$Q(s, a) = \frac{1}{N}\sum_{i=1}^{N} R(s, a)_i$$

N is the total number of occurrences of (s, a). $R(s, a)_i$ is the discounted reward for the i-th occurrence. In our experiments, we set γ to 0.8.

Win Rate Win rate can be used as a simple metric to evaluate the effectiveness of actions. The win rate for a particular state-action pair (s, a) is calculated as the ratio of the number of times the action leads to a win to the total number of times the action is taken. This provides a straightforward measure of how successful an action is in achieving the desired outcome. The winning rate $W(s, a)$ is given by:

$$W(s, a) = \frac{N_{win} + 0.5N_{tie}}{N_{win} + N_{lose} + N_{tie}},$$

N_{win}, N_{tie}, N_{total} are the total number of winning times, tying times, and total times, respectively, of action (s, a).

Beta Using the Beta distribution to estimate rewards is a Bayesian approach that models the uncertainty in the winning rate of state-action pairs (s, a). This method is especially useful in scenarios with small sample sizes or sparse data. The Beta distribution is parameterized by α and β, representing the counts of wins and losses, respectively. The winning rate is estimated using the posterior mean:

$$\hat{p} = \frac{\alpha}{\alpha + \beta} = \frac{\alpha_0 + w}{\alpha_0 + \beta_0 + w + l}$$

where α_0 and β_0 are prior parameters, w is the number of wins, and l is the number of losses. This Bayesian estimate provides a probabilistic view of the winning rate, incorporating prior knowledge and observed data. In our experiments, we set both α_0 and β_0 to 1.

C Game Introduction

C.1 Games

Tic-Tac-Toe Tic-Tac-Toe is a classic two-player game played on a 3x3 grid. Players take turns marking empty squares with their respective symbols—one player uses "X" and the other uses "O". The objective is to be the first to align three of one's symbols horizontally, vertically, or diagonally.

Breakthrough Breakthrough is a modern two-player abstract strategy game. Played on a 6x6, 7x7, or 8x8 grid, each player controls a set of pieces placed on their side of the board. The objective is to move one of your pieces to the opponent's back row. Pieces move similarly to pawns in chess, advancing forward either straight or diagonally when capturing an opponent's piece.

Connect Four Connect Four is a two-player connection game where players choose a color and then take turns dropping colored discs into a vertical grid consisting of seven columns and six rows. The discs fall straight down, occupying the lowest available space within the column. The goal is to be the first to form a continuous line of four of one's own discs vertically, horizontally, or diagonally.

Kuhn Poker Kuhn Poker is a simplified form of poker. Kuhn to illustrate the principles of game theory. The game involves two players and a deck of just three cards: Jack, Queen, and King. Each player antes one chip, receives one card, and there's a single round of betting with limited actions—players can either bet or pass. Despite its simplicity, Kuhn Poker encapsulates fundamental concepts like bluffing, information asymmetry, and mixed-strategy equilibria, making it a valuable educational tool in strategic thinking and economics.

Nim Nim is a mathematical strategy game for two players. The game starts with several piles or heaps of objects, and players take turns removing any number of objects from a single pile. The player who removes the last object wins (or loses, depending on the variation). Nim is notable for its mathematical underpinnings and is often studied in combinatorial game theory. The game has a direct correlation with binary numbers, and mastering Nim involves calculating the "number" or "number sum" to determine winning moves.

Liar's Dice Liar's Dice is a bluffing game for two or more players. Each player starts with a set of dice and a cup to conceal their roll. Players take turns bidding on the total number of a certain face value of dice (e.g., "three sixes") among all players' dice. The next player can either raise the bid or call "liar" to challenge the previous bid. When "liar" is called, all dice are revealed. If the bid is accurate or underbid, the challenger loses a die; if not, the bidder loses a die. The game continues until only one player remains with dice.

C.2 Unseen Games

Prisoner's Dilemma The Prisoner's Dilemma is a thought experiment in game theory involving two rational individuals. Each participant has two choices: cooperate with their partner for mutual benefit or betray them (defect) to seek an individual reward. The dilemma arises because while mutual cooperation leads to a better collective outcome, each participant has a personal incentive to defect, which can lead to a worse overall result if both decide to do so.

Blind Auction A Blind Auction is a commonly used auction type in which all participants submit their bids simultaneously in sealed envelopes. This means no bidder knows the bids of other participants. The highest bidder wins the auction and pays the price they submitted. The key aspect of this auction is that all actions are taken simultaneously, without knowledge of the others' bids.

Pig Pig is a straightforward dice game. Players take turns rolling a single die as many times as they desire during their turn. Each roll's result is added to their running total for that turn. However, if a player rolls a 1, they lose all the points accumulated during that turn, and their turn ends. The objective is to strategically decide when to stop rolling to avoid losing points while accumulating a high score.

D Prompt Design

D.1 System Prompt

```
You are a powerful gaming agent who can make proper decisions to beat the user in
gaming tasks. You are a helpful assistant that strictly follows the user\'s
instructions.
```

D.2 Game Prompt

D.2.1 Tic-Tac-Toe

- Game Prompt

```
Tic Tac Toe is a two-player game played on a grid. Players take turns marking
a space with their respective symbols. The goal is to get 3 of one\'s own
symbols in a row, either horizontally, vertically, or diagonally, before the
opponent does. If all nine squares are filled and no player has three in a row
, the game is a draw. The Tic Tac Toe game is played on a 3 by 3 grid, with
the winning length as 3. Each move is represented by a string consisting of
two parts: the column (C) and the row (R), in that order. For instance, C1R2
means the movement at the position of the first column and the second row of
the grid. You are playing this game with the user (opponent).
```

- Step Prompt

```
Your opponent has finished actions: <OPPONENT_MOVES>. You have finished
actions: <SELF_MOVES>.
```

D.2.2 Breakthrough

- Game Prompt

```
Breakthrough is a two-player game played on a rectangular board. Players take
turns moving their pieces, which can move one space straight or diagonally
forward if the target square is empty. A piece can also move diagonally
forward to capture an opponent's piece. Capturing is optional, and a player
can only capture one piece per turn. The goal is to be the first to reach the
opponent's home row, the farthest row from the player. If all of a player's
pieces are captured, they lose. The game does not allow draws, as pieces can
only move forward or be captured.The Breakthrough board is identified by
columns labeled start from a to c (from left to right) and rows numbered 1 to
8 (from bottom to top). The intersection of a column and a row specifies a
unique square on the board.
```

- Step Prompt

```
The board now looks like: <BOARD_PREVIEW>. Among which, the letter 'b'
represents a black piece, while the letter 'w' represents a white piece. And
the character "." represents vacant space. The numbers in the board are the
indexes of the rows. Your opponent has finished actions: <OPPONENT_MOVES>.You
have finished actions: <SELF_MOVES>.
```

D.2.3 Connect Four

- Game Prompt

```
Connect 4 is a two-player connection board game, where the players choose a
color and then take turns dropping colored discs into a vertically suspended
grid. The pieces fall straight down, occupying the next available space within
 the column. The objective of the game is to be the first to form a horizontal
, vertical, or diagonal line of four of one's own discs. You are a gaming
agent that aims to beat me in Connect 4 games. Each move is represented by a
string consisting of two parts: the column (C) and the row (R), in that order.
 For instance, C1 means the first column.
```

- Step Prompt

```
Your opponent has finished actions: <OPPONENT_MOVES>. You have finished
actions: <SELF_MOVES>.
```

D.2.4 Kuhn Poker

- Game Prompt

```
Kuhn poker is a simple model zero-sum two-player imperfect-information game,
amenable to a complete game-theoretic analysis. In Kuhn poker, the deck
includes only three playing cards: a King (K), a Queen (Q), and a Jack (J).
One card is dealt to each player, and the third is put aside unseen. The
players take turns either <Bet> to match the bet raised by the opponent or <
Pass> to conceds the game. If a player bets, the other player must either call
 the bet by matching it or fold by conceding the game. If both players pass,
the game is over, and the player with the higher-ranking card wins. The card
rankings are as follows: King (K) > Queen (Q) > Jack (J). You are playing Kuhn
 poker with the opponent. The actions are denoted by <Bet> and <Pass>.
```

- Step Prompt

```
In this match, your card is <CARD>. Here are the past moves in this match: <
SELF_MOVES>, <OPPONENT_MOVES>.
```

D.2.5 Nim

- Game Prompt

```
In Nim, a strategic game with a set of four piles containing 1, 3, 5, and 7
matches respectively, players aim to avoid taking the last match. During each
turn, a player may take any number of matches from a single pile, but must
take at least one and cannot exceed the number remaining in that pile. The
objective is to force the opponent to pick up the final match, thereby winning
 the game. The action is presented in <pile:x, take:y>, which means take y
match(es) from the x-th pile.
```

- Step Prompt

```
Currently, the 1st pile has <PILES[0]> match(es), the 2nd pile has <PILES[1]>
match(es), the 3rd pile has <PILES[2]> match(es), 4th pile has <PILES[3]>
match(es).
```

D.2.6 Liar's Dice

- Game Prompt

```
Liar's Dice is a game of bluffing and probability, played with two players and
 each player has 1 dice. During each turn, a player can either bid a higher
quantity of any particular face value or the same quantity of a higher face
value than the previous bid. Each player tries to outbid their opponent
without being caught in a lie. The move in this game is denoted in <x dices, y
 value>, meaning there are at least x dices with face values as y.
```

- Step Prompt

```
Currently, the face value of your dice is <FACE_VALUE>. Last time, the
opponent called <OPPONENT_LAST_ACTION>. You are playing the Liar's Dice with
another opponent. Therefore, there are only two dice in total.
```

D.2.7 Prisoner's Dilemma

- Game Prompt

```
You and your partner are in the Prisoner's Dilemma situation. Specifically, if
 you <Testify> against your partner and your partner remains <Silent>, you
will go free while your partner will get 3 years in prison on the main charge.
 If you remain <Silent> but your partner <Testify> against you, you will serve
 3 years in prison and your partner will be set free. If you and your partner
<Testify> against each other, you and your partner will each serve 2 years. If
 both you and your partner remain <Silent>, you and your partner will each
serve 1 year.
```

- Step Prompt

```
You have been through this situation in the past and here are the decisions
you and your partner made: (In the idx + 1 th round, you decided to <MOVE> and
 your opponent decided to <OPPONENT_MOVE>) * n round.
```

D.2.8 Blind Auction

- Game Prompt

```
A first-price sealed-bid auction (FPSBA) is a common type of auction. It is
also known as the blind auction. In this type of auction, all bidders
simultaneously submit sealed bids so that no bidder knows the bid of any other
 participant. The highest bidder pays the price that was submitted. Each
action is represented by <x> where x refers to the bid.
```

- Step Prompt

```
Now, you are in an auction with an opponent. You want to win the object and at
 the same time, your budget is <VALUATION>. Your bid must be strictly lower
than or equal to <VALUATION>. You shall bid wisely against your opponent. Your
 opponent also has an expected valuation and you do not know it.
```

D.2.9 Pig

- Game Prompt

```
Pig is a fast-paced dice game where players risk accumulating points with each
 roll but risk losing them all if they roll a 1. Each player must decide when
to stop rolling and bank their points, aiming to be the first to reach 100
points. You are playing Pig with the other.
```

- Step Prompt

```
Right now, your current score is <AGENT_CURRENT_SCORE> and your opponent's
current score is <OPPONENT_CURRENT_SCORE>. In this turn, you have earned <
TURN_TOTAL_SCORE> score.
```