

STRATEGIST: SELF-IMPROVEMENT OF LLM DECISION MAKING VIA BI-LEVEL TREE SEARCH

Anonymous authors

Paper under double-blind review

ABSTRACT

Traditional reinforcement learning and planning typically requires vast amounts of data and training to develop effective policies. In contrast, large language models (LLMs) exhibit strong generalization and zero-shot capabilities, but struggle with tasks that require detailed planning and decision-making in complex action spaces. We introduce STRATEGIST, a novel approach that integrates the strengths of both methods. Our approach leverages LLMs to search and update high-level strategy (as text), which are then refined and executed by low-level Monte Carlo Tree Search (MCTS). STRATEGIST is a generalizable framework to optimize the strategy through population-based self-play simulations without the need for any training data. We demonstrate the effectiveness of STRATEGIST in learning optimal strategies for competitive, multi-turn games with partial information, including Game of Pure Strategy (GOPS) and multi-agent, hidden-identity discussion games like The Resistance: Avalon. Our results show that agents equipped with STRATEGIST outperform those trained with traditional RL methods, other LLM-based skill acquisition techniques, pre-existing LLM agents across both game environments, **while achieving comparable performance against human players.**

1 INTRODUCTION

Recent studies have shown the potential of Large Language Models (LLMs) for learning skills and improving decision-making in interactive environments Wang et al. (2022; 2023a; 2024); Xi et al. (2023); Zhao et al. (2024); Liu et al. (2023). However, adversarial, multi-agent environments present a significant challenge. LLMs must reason about opponent actions, plan ahead strategically, and navigate a vast policy space Kambhampati et al. (2024); Valmeekam et al. (2023); Kambhampati (2024). This complexity hinders the LLM’s ability to identify, understand, and translate optimal policies into effective actions. Simply generating decision rules by querying the LLM proves inefficient and impractical in such settings. This raises two critical questions: (1) *How can LLMs effectively learn complex policies in adversarial multi-agent environments?* (2) *What is the best approach to improve these policies?*

This paper introduces STRATEGIST, a novel framework that leverages LLMs to develop high-level strategies, represented as interpretable text, which are then refined and translated into executable policies using a low-level executor (Figure 1). Our approach combines LLM self-improvement with a hierarchical search process, operating on both a high-level strategy space and a low-level action space.

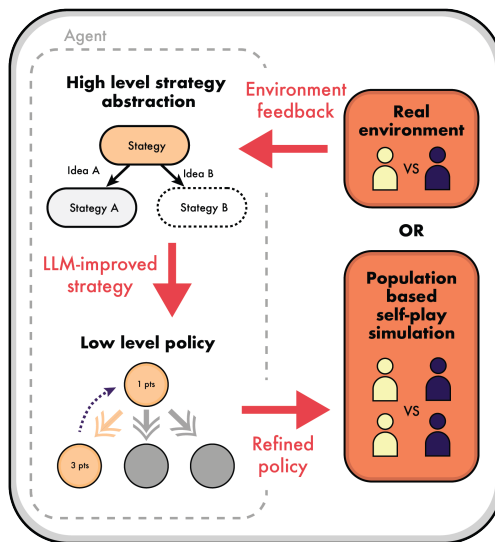


Figure 1: **Overview of STRATEGIST.** We use a LLM to generate and improve high-level strategy abstractions using environment feedback, then refine strategy into a policy using tree search or CoT.

At the high level, STRATEGIST constructs a strategy tree through an evolutionary process, iteratively testing and improving strategies via LLM-guided revisions based on self-play feedback. This process fosters the emergence of robust, adaptable strategies. To further enhance strategy improvement, we maintain a queue of "improvement ideas" sampled via a bandit algorithm to guide LLM revisions. At the low level, we employ fine-grained tree search techniques (e.g., MCTS) guided by the high-level strategy to refine the agent’s policy. We can run the policy against itself through self-play in the multi-agent strategic game, and use win-rate as final outcome reward. The reward can be backpropagated to help update the strategy. This bi-level approach effectively integrates the generalization capabilities of LLMs with the precise planning of MCTS. Both levels are updated through simulated self-play, eliminating the need for extensive training data.

We demonstrate the effectiveness of STRATEGIST in two challenging games: the Game of Pure Strategy (GOPS) and Resistance: Avalon (Ross, 1971; Light et al., 2023), chosen for their strategic complexity and differing levels of natural language interaction. Our results show that STRATEGIST outperforms existing LLM-based self-improvement methods and traditional RL approaches, achieving superior performance within the same computational and data budget. Furthermore, we highlight the successful integration of feedback across the different levels of our framework, demonstrating the synergistic interplay between high-level strategy improvement and low-level policy refinement. **Moreover, STRATEGIST matches human performance in Avalon while employing a more sophisticated mixed strategy and strategic randomization to deceive others and conceal its identity.**

To summarize, our main contributions are:

- We propose STRATEGIST, a general non-parametric bilevel skill-learning framework that uses LLMs to learn high-level strategy abstractions, refined into low-level policies by a modular executor. These abstractions can define strategic profiles for all players in the game.
- We introduce a modular improvement method for high-level strategies using population-based self-play without training data, demonstrating superior performance over existing LLM-based skill learning methods.
- We apply STRATEGIST to GOPS and Avalon, demonstrating its effectiveness in learning policies for both dialogue and non-dialogue actions, achieving higher win rates than both existing agents and human players. Examples of learned strategies are provided in H.3 and I.3.

2 METHODOLOGY

2.1 DECISION MAKING SETTING AND GAMES

The general goal in our decision-making setting is to learn a good policy function in a sequential decision-making setting (generally formulated as a partially observable Markov decision game (POMDG)). Our strategy-learning framework is generalizable to any decision making setting.

Problem definition. Given state space \mathcal{S} and action space \mathcal{A} , a policy function ϕ in policy space Φ is a mapping $\phi : \mathcal{S} \rightarrow \Delta\mathcal{A}$ where we allow ϕ to output a probability distribution over the actions ($\Delta\mathcal{A}$). An environment $\mathcal{E} = \langle \mathcal{S}, \mathcal{A}, \mathcal{N}, T, R, A, \phi_\epsilon \rangle$ defines the state space \mathcal{S} , the action space \mathcal{A} , a set of actors \mathcal{N} , a transition function $T : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$, a reward function $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}^{|\mathcal{N}|}$ that specifies intermediate rewards for each actor, an action function $A : \mathcal{S} \rightarrow \mathcal{N}, \mathcal{P}(\mathcal{A})$ that specifies which actor may take what legal actions at some state where \mathcal{P} is power set, and ϕ_ϵ , the policy function for the environment actor. Note that transitions are deterministic in our notation, and stochastic transitions are handled by the environment actor $\epsilon \in \mathcal{N}$ instead, so we cover stochastic, deterministic, multi-agent, and single agent cases. For a partial information setting, we also have a space of information sets \mathcal{I} and a function $H : \mathcal{S} \times \mathcal{N} \rightarrow \mathcal{I}$ that maps from hidden states and actors to hidden information sets. Hence, $\phi : \mathcal{I} \rightarrow \Delta\mathcal{A}$ is a function from information sets to action distributions instead. A **strategic profile** $\phi = [\phi_i]_{\mathcal{N}}$ describes the policies for every player $i \in \mathcal{N}$.

Let $f : \Sigma \rightarrow \Phi$ be the function that maps strategies to policies. A high-level strategy $\sigma \in \Sigma$ helps parameterize policies so that we can search over the lower dimension Σ space instead of Φ . Let Φ_{-i} denote the space of possible opponent policies, where $-i$ are the indices of players other than i . Then our **goal is to find the optimal strategy** σ_i that approximates finding the optimal policy given the policies of the other agents ϕ_{-i} , i.e.

$$\operatorname{argmax}_{\sigma_i} \mathbb{E}_{\tau \sim (f(\sigma_i), \phi_{-i})} \left[\sum_{(s,a) \in \tau} R_i(s,a) \right] \approx \operatorname{argmax}_{\tau \sim (\phi_i, \phi_{-i})} \mathbb{E} \left[\sum_{(s,a) \in \tau} R_i(s,a) \right]$$

where $\tau = (s_0, a_0, \dots)$ is the simulated trajectory according to the strategic profile (ϕ_i, ϕ_{-i}) and the transition function T , with $a_t \sim \phi(a_t|s_t)$ and $s_{t+1} = T(s_t, a_t)$. Thus, a key part of these settings involves learning the probable strategic profile of other players, usually by assuming that they play optimally like in a Nash equilibrium (Perolat et al., 2022).

The state space, action space, and actor space are different depending on the setting. In non-stochastic, single agent settings such as question answering (Yang et al., 2018; Wang et al., 2022; Shridhar et al., 2020; Thorne et al., 2018), $\mathcal{N} = \{0\}$. In stochastic single agent settings such as web browsing (Yao et al., 2022a; Deng et al., 2024; Zhou et al., 2023), $\mathcal{N} = \{\epsilon, 0\}$ where we add an environment agent ϵ . We specifically focus on adversarial, multi-agent stochastic game ($\mathcal{N} = \{\epsilon, 0, 1, \dots\}$) settings where the other agents are actively working against each other. In non-dialogue-based card games such as GOPS (see B for rules) for example, \mathcal{S} consists of the cards played so far, $\mathcal{N} = \{\epsilon, 0, 1\}$ for players 1 and 2 respectively, and \mathcal{A} consists of the cards you can play. In dialogue-based games such as Avalon (see A for rules), \mathcal{A} consists of both the discrete actions (moves) such as voting, and language actions which consist of text. Similarly, \mathcal{S} consists of both the historical moves and historical dialogue record, and $\mathcal{N} = \{\epsilon, 0, 1, \dots\}$ depending on the number of players involved, with $|\mathcal{N}| \geq 6$.

2.2 ABSTRACTING HIGH LEVEL STRATEGIES AS PROCESSES

A key part of our framework abstracts key features of the policy ϕ , representing them as a high-level strategy σ , which is more suitable for LLM-driven improvement. σ is then executed and refined by a low-level executor, often during inference, resulting in the execution policy ϕ_σ .

feedback from environment \rightarrow LLM-Improver \rightarrow σ \rightarrow Executor(σ) \rightarrow ϕ_σ

Abstraction offers two key benefits: (1) High-level strategies provide a more **abstract, compressed, and intuitive representation** of the problem, which the LLM better understands, enabling its generalization and reasoning capabilities. (2) Searching over high-level strategies is more efficient than low-level policy exploration, as it simplifies the search space by focusing on **core principles rather than fine details**. This approach guides the search toward promising areas more quickly, **leveraging domain knowledge, patterns, or heuristics that generalize across scenarios**.

For **non-dialogue actions**, while the action spaces \mathcal{A} and state spaces \mathcal{S} are typically discrete and finite, the number of possible functions Φ mapping state to action is vast. Most LLM-agents query the LLM directly with state information to decide the next action in decision-making environments (Yao et al., 2023; Shinn et al., 2024; Zhao et al., 2024). However, this is costly because the LLM must be queried for every move, and in Avalon, each player makes at least 20 moves. This becomes even more expensive when incorporating look-ahead search, which requires querying the LLM for future actions and states. Traditionally, policy-gradient reinforcement learning addresses the large policy space by parameterizing the policy and optimizing these parameters, reducing the search space.

We abstract and “parameterize” the policy ϕ as a **value heuristic (VH)** $\sigma := v : \mathcal{S} \rightarrow \mathbb{R}^{|\mathcal{N}|}$ that estimates the expected cumulative returns for each player at a given state, typically the expected probability of winning for each player. Using a value heuristic **simplifies reasoning**, as it’s easier to describe how good a state is than to specify the optimal action. This makes it intuitive for the LLM to reason about winning probabilities. Additionally, we represent the value heuristic as Python code, allowing the LLM to easily process and modify the logic for *computing the value* for each state.

Example LLM Generated Value Heuristic Function

```
def evaluate_state(state):
    # Calculating the potential scores for each player
    player_0_potential_score = sum(state.player_0_hand)
    player_1_potential_score = sum(state.player_1_hand)

    # Calculating the potential final scores for each player
    player_0_final_score = player_0_score + player_0_potential_score
    player_1_final_score = player_1_score + player_1_potential_score

    # Storing the intermediate values used to calculate the scores
    intermediate_values = {
        'player_0_potential_score': player_0_potential_score,
        'player_1_potential_score': player_1_potential_score
    }
    return player_scores, intermediate_values
```

At the low-level, we **execute the strategy** (value function) by selecting the action that leads to the best state: $\phi_i(s) = \operatorname{argmax}_{a \in \mathcal{A}} Q(s, a) = \operatorname{argmax}_{a \in \mathcal{A}} R_i(s, a) + v_i(s') | s' = T(s, a)$. Since

we learn the value v_i for each actor, we can compute the best action for *each actor* i , refining the value function into a strategic profile ϕ . Since value heuristics can be inaccurate, we can refine the policy using search such as MCTS to look ahead through multiple action-state sequences, resulting in a better policy (Grill et al., 2020; Silver et al., 2017; Schrittwieser et al., 2020). MCTS also generates additional feedback by comparing the updated value estimate from MCTS with the initial value heuristic. The estimated win-rate from search provides a **shaped reward signal**, which is more informative than the simple win/lose outcome reward. We focus on key states with the largest discrepancy between the MCTS estimate and the episode outcome for self-improvement feedback. More details on our MCTS implementation are in Appendix F.

Dialogue generation poses a unique challenge for self-improvement. Both the action space \mathcal{A} , representing possible word sequences for each discussion round, and the state space \mathcal{S} , representing prior input dialogues, are enormous. As a result, the number of possible dialogue generation policies Φ is vast, making traditional parameter training approaches difficult to optimize. To address this, we develop a high-level **strategy guide** that the agent processes before speaking. This guide formalizes a **chain-of-thought process** to analyze the situation before generating dialogue (Wei et al., 2022b). It is implemented as a question-and-answer worksheet, where the agent answers all questions in the guide before using it as a prompt for dialogue generation.

Example (fully generated) Chain of Thought Dialogue Strategy Guide

Q1: Who are the players you suspect are on the Evil side and why?
A1: I suspect Player 2 and Player 4 are on the Evil side. Player 2’s suggestion for the team consisting of themselves and Player 1 seems too convenient, as it would allow Evil to easily fail the quest. Player 4’s agreement with Player 1 and Player 2’s proposed team without offering any alternative suggestions raises suspicion.

Q2: Draft a subtle statement that casts doubt on these players without directly accusing them.
A2: "I can’t help but wonder if some players are pushing for certain team compositions to benefit themselves rather than considering what is truly best for the success of our Quest."

Generated dialogue: "I have observed patterns in the voting that suggest certain players may be working together to push for specific team compositions. This raises suspicion and calls into question their true intentions. We must not allow ourselves to be swayed by potentially deceitful tactics."

2.3 MODULAR SEARCH FOR HIGH-LEVEL STRATEGY IMPROVEMENT

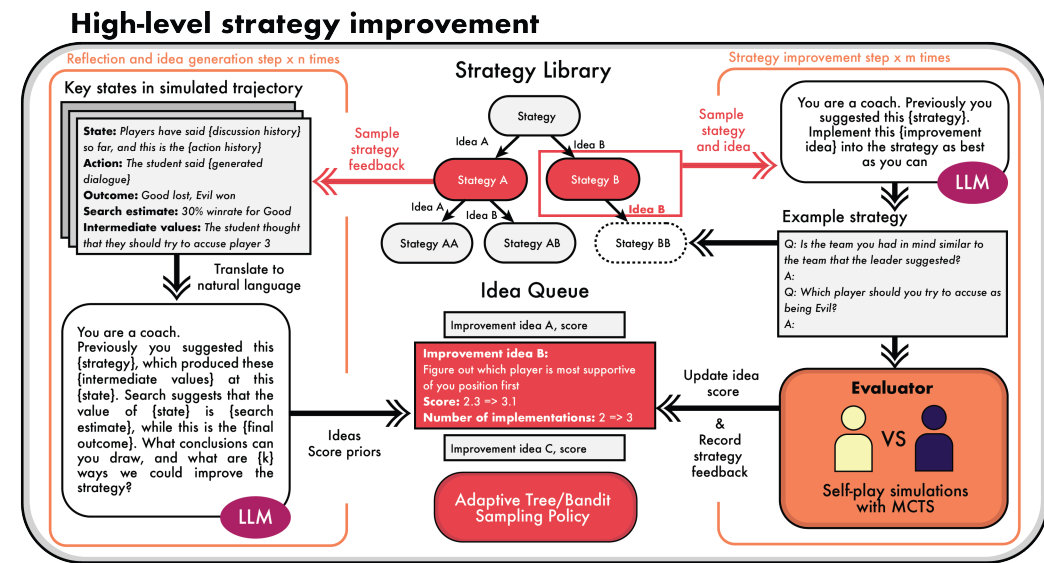


Figure 2: **Overview of our high-level strategy improvement method:** The process alternates between two steps—idea generation and strategy improvement. For more details see Figure 11.

Our framework alternates between two steps: **idea generation** and **strategy improvement**, as shown in Figure 2 and detailed in App. D. Each cycle refines strategies stored in a tree structure based on feedback from self-play simulations and generates new improvement ideas stored in a priority queue.

2.3.1 IDEA GENERATION

In the **idea generation step**, a strategy σ and its feedback trajectory τ_σ are selected from the strategy tree using an adaptive selection policy (e.g., UCB or BFS). Feedback consists of trajectories from previous self-play simulations, including visited states, actions taken, estimated win rates, final outcomes, and intermediate values. To avoid processing lengthy trajectories, we select key states that best capture discrepancies between the strategy’s value heuristic and search-based estimates.

These key states are translated into natural language and used to prompt the LLM for new improvement ideas. The ideas are added to the idea queue with a prior score estimating their potential effectiveness:

$$\text{Idea-queue} \leftarrow \text{LLM-idea-inventor}(\text{Sampled strategy, strategy feedback})$$

2.3.2 STRATEGY IMPROVEMENT

In the **strategy improvement step**, we sample a strategy σ from the strategy tree and an idea d from the queue, using a method that balances exploration and exploitation (e.g., UCB). The LLM refines σ using d , generating a new strategy σ_{new} . This new strategy is evaluated via self-play simulations, which produce win rates $W[\sigma]$ and trajectory feedback $\mathcal{T}[\sigma]$.

During simulations, players conduct MCTS tree searches to estimate win rates at different states, providing additional feedback. The strategy tree is updated with the new strategy and its performance:

$$\text{Strategy-library} \leftarrow \text{LLM-reviser}(\text{Sampled idea, sampled prior strategy})$$

The improvement score of the idea d is updated based on how much it improved σ .

2.3.3 SEARCH MODULARIZATION

A key feature of our framework is the use of an *idea queue* to modularize the search process. By refining strategies incrementally rather than globally, we avoid confounding factors and ensure interpretability of changes. The queue also tracks successful improvements that are generalizable across strategies, enabling transfer and reuse of ideas. Improvements are often additive (e.g., penalties or adjustments) and enhance performance when applied to similar strategies (Table 2).

We use UCB sampling to balance exploration of new ideas and exploitation of proven ones:

$$UCB(\text{idea}) = \bar{z}_{\text{idea}} + c \sqrt{\frac{\ln(N_{\text{total}})}{N_{\text{idea}}}}$$

where \bar{z} is the empirical average improvement score, N_{total} is the total number of ideas implemented, and N_{idea} is the number of implementations of the specific idea.

To simplify the improvement process, we optimize dialogue generation and gameplay moves separately in Avalon before integrating them into a unified agent (Appendix E). This modular approach mirrors strategies used in related domains, such as Diplomacy (FAIR).

2.4 POPULATION BASED SELF-PLAY SIMULATION

Recent works focus on using either another LLM for critique-based feedback (Madaan et al., 2024; Shinn et al., 2024), real environment interactions (Nottingham et al., 2024), or a combination of both (Wang et al., 2023a) during the LLM-improvement process. Our method improves on this by simulating opponents using the learned high-level strategy, enabling higher-quality feedback. Since our high-level strategies can be refined into a policy for any player, we can easily pit different strategies against each other by refining them into specific policies for different players. Specifically, we use **population based self-play**, where we conduct round-robin games among the top ten strategies generated and use the average performance as the strategy score for each strategy (Xu et al., 2023b). During round-robin games, strategies are given the same level of low-level refinement to ensure fairness. This evolutionary approach makes the final improved strategy more robust to different opponent policies, since it will have survived multiple round-robin tournaments. We demonstrate the effectiveness of this method in section 3.5.

3 EXPERIMENTS

We evaluate the effectiveness of our bi-level framework through four key experiments: demonstrating STRATEGIST’s competitiveness with human players, its superiority over RL-based deep learning methods, its advantage over LLM-based agents, and the effectiveness of our LLM-improvement method compared to other techniques. Our approach is tested on two games with distinct challenges. **Game of Pure Strategy (GOPS)** is a two-player, zero-sum card game requiring strategic decisions under partial information and simultaneous moves (see B) (Ross, 1971; Lanctot et al., 2009). **The Resistance: Avalon** is a multi-agent (5+ players), team-based game combining language, deception, and deduction (see A) (Light et al., 2023). These games highlight STRATEGIST’s versatility in handling strategic and social complexities.

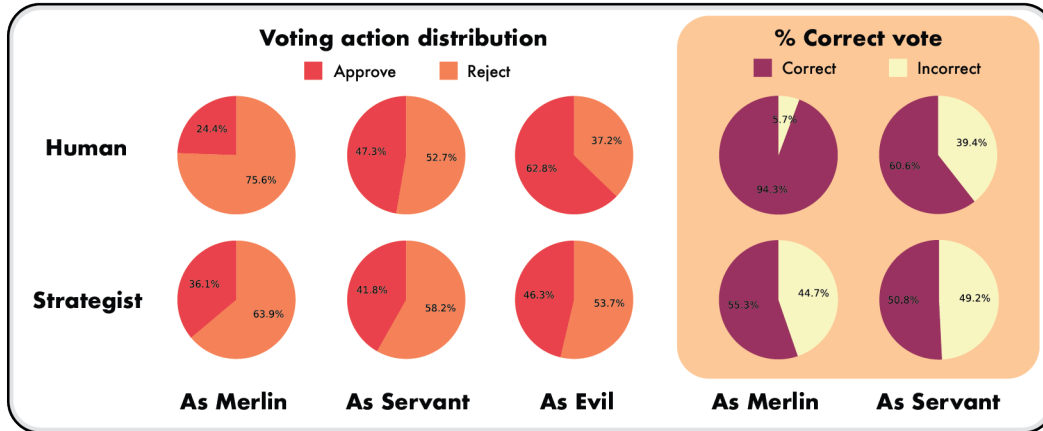


Figure 3: **Action analysis of Strategist and humans when playing against each other in Avalon.** Strategist conducts more strategic randomization than humans, making it harder to deduce their identity based on their actions than humans.

3.1 HUMAN EVALUATIONS

We conducted human experiments in the six-player Avalon setting, using the following character roles: Merlin, 3 Servants of Arthur, 1 Assassin, and 1 Minion of Mordred. Ten participants were recruited and randomly assigned to one of the roles, playing against five STRATEGIST agents. In total, 30 games were collected and analyzed. As presented in Table 1, STRATEGIST achieved a win rate comparable to human players. After each session, participants completed a survey evaluating the performance of STRATEGIST across seven key metrics, while an experimenter independently assessed human performance. The survey results are visualized in Figure 4, revealing that STRATEGIST outperformed humans in concealment and adaptability to human playstyles, but lagged in reasoning, deduction, and cooperation.

Additionally, we conducted an **action analysis** to compare the behavior of STRATEGIST and human players, shown in Figure 3. Specifically, we examined the distribution of approval votes—a pivotal action in Avalon. The data revealed that human players exhibited distinct voting patterns based on their roles, facilitating the deduction of their identities. In contrast, STRATEGIST demonstrated **strategic randomization**, producing more uniform action distributions across roles and thereby making its identity harder to infer. More details on human evaluations can be found in Appendix O.

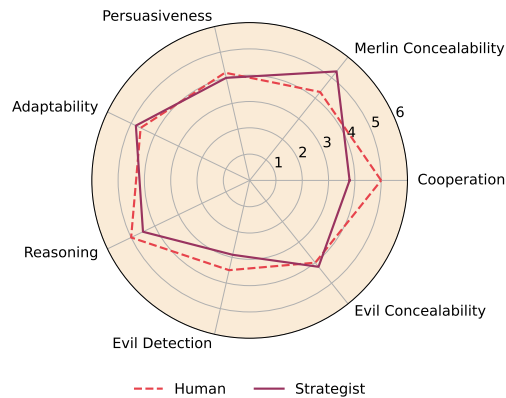


Figure 4: **Human vs STRATEGIST performance metrics in Avalon.** We ask humans to evaluate the performance of STRATEGIST and human players in ad-hoc human-AI games. STRATEGIST performs better than humans at concealment, but worse at reasoning and cooperation.

Table 2: **Comparison of different self-improvement methods.** The improvement process was run with the same number of strategies generated for each method (40 for GOPS, 24 for Avalon) on the same seed functions, all with GPT3.5. We collect 9 functions from each process and play them against each other (a total of $5 \times 9 = 40$ different agents), reporting the average number of points you win over your opponents for GOPS and the average winrate for Avalon. For the dialogue guide, we show the improvement over the baseline seed function, which has a score $z = -0.875 \in [-2, 2]$, a rating from opponents which we describe in G.

Self-Improvement Methodology	Line search (Madaan et al., 2024; Shinn et al., 2024)	Greedy search (Ma et al., 2023)	Best First Search (BFS) (Yao et al., 2024)	BFS with thought	STRATEGIST
GOPS Value Heuristic	-0.47 ± 0.74	-0.54 ± 0.45	0.092 ± 0.67	-0.48 ± 0.375	1.5 ± 0.99
Avalon Value Heuristic	0.54 ± 0.11	0.47 ± 0.11	0.50 ± 0.085	0.55 ± 0.065	0.59 ± 0.11
Merlin Dialogue Guide	0.37 ± 0.19	0.62 ± 0.13	0.49 ± 0.063	0.37 ± 0.06	0.88 ± 0.063
Assassin Dialogue Guide	1.83 ± 0.25	1.81 ± 0.063	1.82 ± 0.063	1.89 ± 0.063	1.98 ± 0.042

We further analyzed the proportion of **correct votes**, defined as rejecting teams with at least one Evil player or approving teams with only Good players. This metric indicates a player’s ability to infer Good and Evil identities, a skill particularly crucial for the Merlin role. As shown in Figure 3, STRATEGIST employed increased randomization as Merlin, minimizing information leakage through voting patterns. Conversely, human players displayed highly role-specific correctness trends. These findings align with survey feedback, corroborating that STRATEGIST excels in concealment strategies compared to human counterparts.

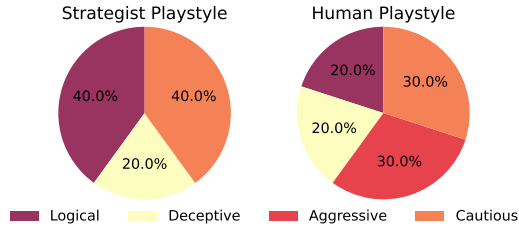


Figure 5: Description of STRATEGIST play-style compared to human players, as evaluated by human participants. STRATEGIST is described as being more logical and cautious.

3.2 DIFFERENT LLM IMPROVEMENT METHODS

We demonstrate the effectiveness of our strategy improvement method by benchmarking it against four other skill-improvement methods. **Line search** (Madaan et al., 2024) always reflects and improves upon the latest improved strategy. **Greedy search** (Ma et al., 2023) selects the best strategy from the last generation of improved strategies to improve upon each improvement cycle. **Best first search** (Yao et al., 2024) improves upon the k best strategies generated in any iteration of each improvement cycle. **Best first search with thought** enhances BFS by prompting the LLM to reflect, think, and propose ideas on how to improve the previous strategy before improving the strategy itself (Yao et al., 2022b). STRATEGIST is our method that uses an additional idea queue Q and an idea generation step to guide the improvement process. Comparing BFS with thought and STRATEGIST helps demonstrate the value of having a modularized reflection and idea proposition process that is separate from strategy generation. More details can be found in Appendix N.

Table 1: Results of STRATEGIST vs Human.

Metric	Human	STRATEGIST
Winrate	0.367	0.333
Standard Error	0.089	0.061

Our results, presented in Table 2, use the same feedback method (simulation-based population self-play) while varying the improvement methods, with a constant number of new strategies generated across all methods. Figure 12 (right) shows the gameplay scores of these strategies on GOPS. Even when controlling for the number of output tokens generated by the LLM, our method outperforms the others, as demonstrated in Figure 18. We attribute this to (1) the idea queue, which helps test incremental improvements and guide the search process, and (2) our strategy and idea selection policy, which more efficiently explores the strategy space and escapes local maxima (Figure 12 left).

3.3 INTERACTION BETWEEN LOW-LEVEL REFINEMENT AND HIGH LEVEL STRATEGY SEARCH

378 We demonstrate that both our high-level
 379 strategy improvement and low-level refine-
 380 ment processes scale effectively with in-
 381 creased budgets by showing how differ-
 382 ent high-level strategies performance with
 383 varying MCTS search budgets. A higher
 384 search budget allows for more nodes to
 385 be looked ahead before taking actions,
 386 resulting in a more refined policy. As
 387 shown in Figure 6, strategies improved
 388 with more iterations using STRATEGIST ex-
 389 hibit both higher performance and greater
 390 performance gains with increased refine-
 391 ment. Policy refinement also scales excep-
 392 tionally well.

393 3.4 LLM-IMPROVEMENT 394 VS. REINFORCEMENT 395 LEARNING (RL) BASED METHODS

397 We demonstrate the effectiveness of our
 398 method compared to traditional RL-based
 399 approaches for learning a good policy.
 400 Specifically, we show that our method
 401 learns a value heuristic function more effi-
 402 ciently than deep RL, as used in AlphaGo,
 403 MuZero (Silver et al., 2017; Schrittwieser
 404 et al., 2020), and DeepRole (Serrino et al.,
 405 2019). While AlphaGo uses MCTS (Koc-
 406 sis & Szepesvári, 2006) with a deep value
 407 network, DeepRole combines counterfactual
 408 regret minimization with a deep value
 409 network (Moravčík et al., 2017; Zinkevich et al., 2007). We adapt both algorithms to our setting.

409 Although deep RL can approximate the true value function with enough data, training, and a large
 410 network, we ensure a fair comparison by (1) limiting both RL methods and our approach to the
 411 same number of simulated episodes, and (2) capping the number of training steps after each batch of
 412 episode trajectory data. While both RL methods and STRATEGIST use the same number of self-play
 413 episodes, STRATEGIST only trains on a small subset of transition steps, whereas RL methods train
 414 on the entire dataset. Our results, shown in Table 3 and Figure 7, demonstrate that STRATEGIST
 415 consistently outperforms both AlphaGo and DeepRole. More details on our RL implementation and
 416 the effects of increased sample size can be found in Appendix L, where we observe performance
 417 improvements with additional samples.

418 **Table 3: Comparison of reinforcement learning vs our method.** We run each process 10 times,
 419 taking the best strategy generated by each run and playing them against each other for 1024 games.
 420 Average win-rates, total number of self-play episodes, and training transition steps across best
 421 generated strategies are shown here.

Setting	Metric	VS Alpha-go		VS DeepRole	
		Alpha-go	STRATEGIST	DeepRole	STRATEGIST
GOPS	Point difference	-0.39 ± 0.22	0.33 ± 0.38	-0.56 ± 0.18	1.14 ± 0.09
	Self-play eps.	320 episodes	320 episodes	320 episodes	320 episodes
	Trans. steps	$\sim 3,840$ steps	100 steps	$\sim 3,840$ steps	100 steps
Avalon	Winrate	0.30 ± 0.03	0.38 ± 0.04	0.33 ± 0.04	0.44 ± 0.02
	Self-play eps.	160 episodes	160 episodes	160 episodes	160 episodes
	Trans. steps	$\sim 24,000$ steps	60 steps	$\sim 24,000$ steps	60 steps

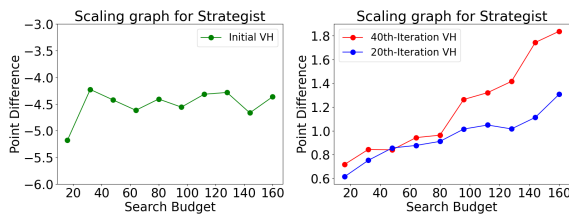


Figure 6: **Performance of high level strategies on GOPS as we vary the level of low-level MCTS refinement (MCTS search budget).** *Left:* Scaling curve for initial unimproved LLM-generated value-heuristic. *Right:* Scaling curve for improved value-heuristic using STRATEGIST after 20 and 40 iterations (steps) of self-improvement respectively. STRATEGIST improved value heuristics scale better with increased low-level refinement.

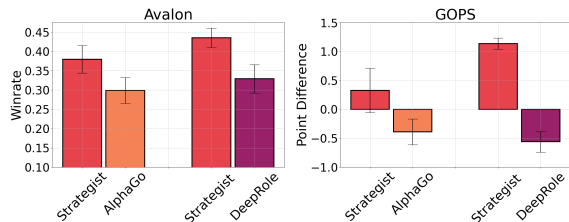


Figure 7: **Performance of different training methods against the baseline in Avalon and GOPS.**

3.5 FEEDBACK QUALITY AND REWARD SIGNAL

We benchmark our feedback method (population based self-play) against (1) an **LLM-critic** (Madaan et al., 2024) and (2) trajectory feedback from a **fixed opponent policy**, with results in Table 4 showing superior performance in both GOPS action planning and dialogue generation. This shows the effectiveness of our evolutionary population based self-play approach, and that of our strategy abstraction that learns a strategic profile for all players.

Table 4: **Comparison of different methods of collecting feedback.** All methods use the same high-level improvement process (STRATEGIST). For GOPS we collect 24 generated functions from each method and play them against each other. For Avalon we evaluate 9 generated guides. We simulated 32 games for GOPS and 4 rounds of discussion for Avalon to collect gameplay feedback during each improvement step, across 10 and 6 improvement rounds for GOPS and Avalon respectively.

Setting/Method	Metric	LLM-critic	Fixed opponent	Population-based Self-play
GOPS	Point difference	-0.27 ± 1.1	0.089 ± 0.86	0.87 ± 1.5
	# of opponents	0	1	4
Avalon	Winrate	0.37 ± 0.063	0.62 ± 0.13	0.88 ± 0.06
	# of opponents	0	1	4

3.6 STRATEGIST VS. LLM AGENTS

We demonstrate that our method also achieves better performance against other LLM-agents. ReAct is a popular LLM-agent that prompts the LLM to think before taking an action (Yao et al., 2022b). ReCon is a LLM-agent for hidden identity games that prompts the LLM to recursively contemplate on what opponents are thinking and might do before taking actions (Wang et al., 2023b). We adapt ReCon to our setting by asking the agent to recursively contemplate. Our gameplay results are shown in Table 5. This suggests that through STRATEGIST, our LLM-agent is able to learn high-level strategies similar in performance to those of ReCon, such as recursive contemplation.

Table 5: Results of STRATEGIST playing against LLM-based baselines, i.e., ReAct and ReCon.

Metric	VS ReAct		VS ReCon	
	ReAct	STRATEGIST	ReCon	STRATEGIST
Winrate	47.5 ± 2.5	52.5 ± 2.5	38.9 ± 5.5	61.1 ± 5.5
#Tokens per round	56 ± 14.3	164.3 ± 27.7	245.7 ± 21.2	248.2 ± 24.1

4 RELATED WORK

LLMs for Text Agents. Large language models (LLMs) demonstrate emergent capabilities such as zero-shot prompting, reasoning, and extensive world knowledge (Bommasani et al., 2021; Brown et al., 2020; Wei et al., 2022a; Yu et al., 2023a). Recent frameworks like ReAct (Yao et al., 2023) and Reflexion (Shinn et al., 2024) have incorporated reasoning, memory, feedback, and tool use to improve decision-making (Wang et al., 2023a; Huang et al., 2022; Schick et al., 2024). Prompting techniques like Chain-of-Thought and Tree-of-Thought are effective for reasoning (Wei et al., 2022b; Yao et al., 2024) but fall short in complex games requiring iterative self-improvement. Our method, STRATEGIST, introduces a bi-level tree search combining high-level planning and self-play for feedback-driven strategy refinement.

Skill Learning with LLMs. LLMs have been used to acquire skills by learning textual memories or insights (Shinn et al., 2024; Zhao et al., 2024). However, textual approaches struggle with long, quantitative trajectories. We focus on high-level strategies optimized through simulational self-play, enabling robust skill learning in multi-agent environments. While reward models learned by LLMs

excel in single-agent tasks (Ma et al., 2023; Yu et al., 2023b), we extend these ideas to multi-agent settings by introducing methods for feedback generation and strategy improvement.

AI in Strategy Games. Advances like AlphaGo and MuZero highlight the synergy of MCTS, deep learning, and self-play (Silver et al., 2017; Schrittwieser et al., 2020). LLMs have also been integrated into dialogue-based games such as Diplomacy (, FAIR) and Texas Hold'em (Zhang et al., 2024). Our approach bridges these domains by enabling LLMs to both train value heuristics more efficiently than RL and generate dialogue for social deduction games without human examples. Additionally, our method has potential applications in negotiation tasks (Abdelnabi et al., 2023; Fu et al., 2023).

LLM-agents for discussion games. There has been much recent interest in ... since discussion games provide an excellent benchmark for accessing the reasoning and planning abilities of LLMs. This includes Werewolf (Bailis et al., 2024; Xu et al., 2023b;a; Wu et al., 2024), another social deduction game, and other negotiation and word games (Fu et al., 2023; Cheng et al., 2024). We give a more detailed comparison of our work to other AI agents in Table 6 and App. P.

Metric	EnReaWolf	Cicero	ComWolf	LARLWolf	SPAG	AlphaGo	ICL-AIF	Agent-Pro	ReCon	DeepRole	Strategist
Used in Avalon	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓
Parameter-free	✗	✗	✓	✗	✗	✗	✓	✓	✓	✗	✓
Uses LM	✓	✓	✓	✓	✓	✗	✓	✓	✓	✗	✓
Self-improve	✓	✓	✗	✓	✓	✓	✓	✓	✗	✓	✓
Tree Search	✗	✓	✗	✗	✗	✓	✗	✗	✗	✓	✓
Human-annotation free	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓
Belief updates	✗	✗	✗	✗	✗	✗	✗	✓	✗	✓	✓
Bi-level improvement	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓

Table 6: Comparison of Strategist with Related Works. We use an bi-level improvement approach, paired with an advanced high-level strategy learning process that is parameter free. Our approach handles beliefs and partial observability and uses low-level policy refinement (tree-search) to further enhance our policy. For details see App. P

5 CONCLUSION

In conclusion, we have introduced STRATEGIST, a bi-level framework that facilitates high-level abstract strategy learning with LLMs and a generalizable non-parametric self-improvement mechanism. This approach enables the model to learn and refine skills autonomously. Without relying on task-specific prompts or human-generated policy data, our method demonstrates its ability to learn effective strategies through population-based self-play, guided solely by the rules of the game.

The performance of STRATEGIST highlights the potential of leveraging LLMs for generating high-level strategic abstractions, while using low-level tree search to iteratively refine the policy. This dual approach not only showcases the power of modular guidance—whether through high-level strategy exploration or low-level self-play feedback—but also underlines the value of integrating these processes for accelerated and robust skill acquisition in LLMs.

Furthermore, our results suggest broader implications for the development of LLM-driven decision-making agents across a range of complex domains. By embedding flexible and adaptive learning mechanisms, our framework paves the way for more autonomous systems capable of mastering tasks with minimal human intervention. This opens new avenues for the application of LLMs in areas such as multi-agent systems, reinforcement learning, and general AI, where strategic planning and self-improvement are key.

540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593

REFERENCES

- Sahar Abdelnabi, Amr Gomaa, Sarath Sivaprasad, Lea Schönherr, and Mario Fritz. Llm-deliberation: Evaluating llms with interactive multi-agent negotiation games. *arXiv preprint arXiv:2309.17234*, 2023.
- Suma Bailis, Jane Friedhoff, and Feiyang Chen. Werewolf arena: A case study in llm evaluation via social deduction. *arXiv preprint arXiv:2407.13943*, 2024.
- Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, Shyamal Buch, Dallas Card, Rodrigo Castellon, Niladri Chatterji, Annie Chen, Kathleen Creel, Jared Quincy Davis, Dora Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren Gillespie, Karan Goel, Noah Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, Omar Khattab, Pang Wei Koh, Mark Krass, Ranjay Krishna, Rohith Kuditipudi, Ananya Kumar, Faisal Ladhak, Mina Lee, Tony Lee, Jure Leskovec, Isabelle Levent, Xiang Lisa Li, Xuechen Li, Tengyu Ma, Ali Malik, Christopher D. Manning, Suvir Mirchandani, Eric Mitchell, Zanele Munyikwa, Suraj Nair, Avani Narayan, Deepak Narayanan, Ben Newman, Allen Nie, Juan Carlos Niebles, Hamed Nilforoshan, Julian Nyarko, Giray Ogut, Laurel Orr, Isabel Papadimitriou, Joon Sung Park, Chris Piech, Eva Portelance, Christopher Potts, Aditi Raghunathan, Rob Reich, Hongyu Ren, Frieda Rong, Yusuf Roohani, Camilo Ruiz, Jack Ryan, Christopher Ré, Dorsa Sadigh, Shiori Sagawa, Keshav Santhanam, Andy Shih, Krishnan Srinivasan, Alex Tamkin, Rohan Taori, Armin W. Thomas, Florian Tramèr, Rose E. Wang, William Wang, Bohan Wu, Jiajun Wu, Yuhuai Wu, Sang Michael Xie, Michihiro Yasunaga, Jiaxuan You, Matei Zaharia, Michael Zhang, Tianyi Zhang, Xikun Zhang, Yuhui Zhang, Lucia Zheng, Kaitlyn Zhou, and Percy Liang. On the opportunities and risks of foundation models. *arXiv preprint arXiv: Arxiv-2108.07258*, 2021.
- Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In Hugo Larochelle, Marc’Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL <https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html>.
- Pengyu Cheng, Tianhao Hu, Han Xu, Zhisong Zhang, Yong Dai, Lei Han, and Nan Du. Self-playing adversarial language game enhances llm reasoning. *arXiv preprint arXiv:2404.10642*, 2024.
- Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems*, 36, 2024.
- Meta Fundamental AI Research Diplomacy Team (FAIR)[†], Anton Bakhtin, Noam Brown, Emily Dinan, Gabriele Farina, Colin Flaherty, Daniel Fried, Andrew Goff, Jonathan Gray, Hengyuan Hu, et al. Human-level play in the game of diplomacy by combining language models with strategic reasoning. *Science*, 378(6624):1067–1074, 2022.
- Yao Fu, Hao Peng, Tushar Khot, and Mirella Lapata. Improving language model negotiation with self-play and in-context learning from ai feedback. *arXiv preprint arXiv:2305.10142*, 2023.
- Jean-Bastien Grill, Florent Althé, Yunhao Tang, Thomas Hubert, Michal Valko, Ioannis Antonoglou, and Rémi Munos. Monte-carlo tree search as regularized policy optimization. In *International Conference on Machine Learning*, pp. 3769–3778. PMLR, 2020.

594 Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan
595 Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Noah Brown, Tomas Jackson, Linda
596 Luu, Sergey Levine, Karol Hausman, and Brian Ichter. Inner monologue: Embodied reasoning
597 through planning with language models. *arXiv preprint arXiv: Arxiv-2207.05608*, 2022.

598
599 Subbarao Kambhampati. Can large language models reason and plan? *Annals of the New York
600 Academy of Sciences*, 1534(1):15–18, 2024.

601 Subbarao Kambhampati, Karthik Valmeekam, Lin Guan, Kaya Stechly, Mudit Verma, Siddhant
602 Bhambri, Lucas Saldyt, and Anil Murthy. Llms can’t plan, but can help planning in llm-modulo
603 frameworks. *arXiv preprint arXiv:2402.01817*, 2024.

604
605 Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European conference
606 on machine learning*, pp. 282–293. Springer, 2006.

607 Marc Lanctot, Kevin Waugh, Martin Zinkevich, and Michael Bowling. Monte carlo sampling for
608 regret minimization in extensive games. *Advances in neural information processing systems*, 22,
609 2009.

610
611 Jonathan Light, Min Cai, Sheng Shen, and Ziniu Hu. From text to tactic: Evaluating llms playing the
612 game of avalon. *arXiv preprint arXiv:2310.05036*, 2023.

613
614 Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding,
615 Kaiwen Men, Kejuan Yang, et al. Agentbench: Evaluating llms as agents. *arXiv preprint
616 arXiv:2308.03688*, 2023.

617
618 Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman,
619 Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding
620 large language models. *arXiv preprint arXiv:2310.12931*, 2023.

621
622 Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegrefe, Uri
623 Alon, Nouha Dziri, Shrimai Prabhunoye, Yiming Yang, et al. Self-refine: Iterative refinement
624 with self-feedback. *Advances in Neural Information Processing Systems*, 36, 2024.

625
626 Matej Moravčík, Martin Schmid, Neil Burch, Viliam Lisý, Dustin Morrill, Nolan Bard, Trevor
627 Davis, Kevin Waugh, Michael Johanson, and Michael Bowling. Deepstack: Expert-level artificial
628 intelligence in heads-up no-limit poker. *Science*, 356(6337):508–513, 2017.

629
630 Kolby Nottingham, Bodhisattwa Prasad Majumder, Bhavana Dalvi Mishra, Sameer Singh, Peter
631 Clark, and Roy Fox. Skill set optimization: Reinforcing language model behavior via transferable
632 skills. *arXiv preprint arXiv:2402.03244*, 2024.

633
634 Julien Perolat, Bart De Vylder, Daniel Hennes, Eugene Tarassov, Florian Strub, Vincent de Boer,
635 Paul Muller, Jerome T Connor, Neil Burch, Thomas Anthony, et al. Mastering the game of stratego
636 with model-free multiagent reinforcement learning. *Science*, 378(6623):990–996, 2022.

637
638 Sheldon M Ross. Goofspiel—the game of pure strategy. *Journal of Applied Probability*, 8(3):
639 621–625, 1971.

640
641 Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke
642 Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach
643 themselves to use tools. *Advances in Neural Information Processing Systems*, 36, 2024.

644
645 Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon
646 Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari,
647 go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

648
649 Jack Serrino, Max Kleiman-Weiner, David C Parkes, and Josh Tenenbaum. Finding friend and foe in
650 multi-agent games. *Advances in Neural Information Processing Systems*, 32, 2019.

651
652 Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion:
653 Language agents with verbal reinforcement learning. *Advances in Neural Information Processing
654 Systems*, 36, 2024.

648 Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew
649 Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv*
650 *preprint arXiv:2010.03768*, 2020.
651

652 David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez,
653 Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without
654 human knowledge. *nature*, 550(7676):354–359, 2017.

655 Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.
656

657 James Thorne, Andreas Vlachos, Christos Christodoulopoulos, and Arpit Mittal. Fever: a large-scale
658 dataset for fact extraction and verification. *arXiv preprint arXiv:1803.05355*, 2018.

659 Karthik Valmeekam, Matthew Marquez, Sarath Sreedharan, and Subbarao Kambhampati. On the
660 planning abilities of large language models—a critical investigation. *Advances in Neural Information*
661 *Processing Systems*, 36:75993–76005, 2023.
662

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

666 Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai
667 Tang, Xu Chen, Yankai Lin, et al. A survey on large language model based autonomous agents.
668 *Frontiers of Computer Science*, 18(6):186345, 2024.
669

670 Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. Scienceworld: Is
671 your agent smarter than a 5th grader? *arXiv preprint arXiv:2203.07540*, 2022.

672 Shenzhi Wang, Chang Liu, Zilong Zheng, Siyuan Qi, Shuo Chen, Qisen Yang, Andrew Zhao, Chaofei
673 Wang, Shiji Song, and Gao Huang. Avalon’s game of thoughts: Battle against deception through
674 recursive contemplation. *arXiv preprint arXiv:2310.01320*, 2023b.
675

676 Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama,
677 Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals,
678 Percy Liang, Jeff Dean, and William Fedus. Emergent abilities of large language models. *arXiv*
679 *preprint arXiv: Arxiv-2206.07682*, 2022a.

680 Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny
681 Zhou. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint*
682 *arXiv:2201.11903*, 2022b.

683 Wikipedia contributors. Goofspiel — Wikipedia, the free encyclopedia, 2023. URL [https://en.
684 wikipedia.org/w/index.php?title=Goofspiel&oldid=1174471596](https://en.wikipedia.org/w/index.php?title=Goofspiel&oldid=1174471596). [Online;
685 accessed 22-May-2024].
686

687 Shuang Wu, Liwen Zhu, Tao Yang, Shiwei Xu, Qiang Fu, Yang Wei, and Haobo Fu. Enhance
688 reasoning for large language models in the game werewolf. *arXiv preprint arXiv:2402.02330*,
689 2024.

690 Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe
691 Wang, Senjie Jin, Enyu Zhou, et al. The rise and potential of large language model based agents:
692 A survey. *arXiv preprint arXiv:2309.07864*, 2023.
693

694 Yuzhuang Xu, Shuo Wang, Peng Li, Fuwen Luo, Xiaolong Wang, Weidong Liu, and Yang Liu.
695 Exploring large language models for communication games: An empirical study on werewolf.
696 *arXiv preprint arXiv:2309.04658*, 2023a.

697 Zelai Xu, Chao Yu, Fei Fang, Yu Wang, and Yi Wu. Language agents with reinforcement learning for
698 strategic play in the werewolf game. *arXiv preprint arXiv:2310.18940*, 2023b.
699

700 Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W Cohen, Ruslan Salakhutdinov,
701 and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question
answering. *arXiv preprint arXiv:1809.09600*, 2018.

702 Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable
703 real-world web interaction with grounded language agents. *Advances in Neural Information*
704 *Processing Systems*, 35:20744–20757, 2022a.

705
706 Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao.
707 React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*,
708 2022b.

709
710 Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao.
711 React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*,
712 2023.

713
714 Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan.
715 Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural*
716 *Information Processing Systems*, 36, 2024.

717
718 Jifan Yu, Xiaozhi Wang, Shangqing Tu, Shulin Cao, Daniel Zhang-Li, Xin Lv, Hao Peng, Zijun
719 Yao, Xiaohan Zhang, Hanming Li, et al. Kola: Carefully benchmarking world knowledge of large
720 language models. *arXiv preprint arXiv:2306.09296*, 2023a.

721
722 Wenhao Yu, Nimrod Gileadi, Chuyuan Fu, Sean Kirmani, Kuang-Huei Lee, Montse Gonzalez Arenas,
723 Hao-Tien Lewis Chiang, Tom Erez, Leonard Hasenclever, Jan Humplik, et al. Language to rewards
724 for robotic skill synthesis. *arXiv preprint arXiv:2306.08647*, 2023b.

725
726 Wenqi Zhang, Ke Tang, Hai Wu, Mengna Wang, Yongliang Shen, Guiyang Hou, Zeqi Tan, Peng Li,
727 Yueting Zhuang, and Weiming Lu. Agent-pro: Learning to evolve via policy-level reflection and
728 optimization. *arXiv preprint arXiv:2402.17574*, 2024.

729
730 Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm
731 agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*,
732 volume 38, pp. 19632–19642, 2024.

733
734 Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng,
735 Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building
736 autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023.

737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755

A RESISTANCE: AVALON GAME DESCRIPTION

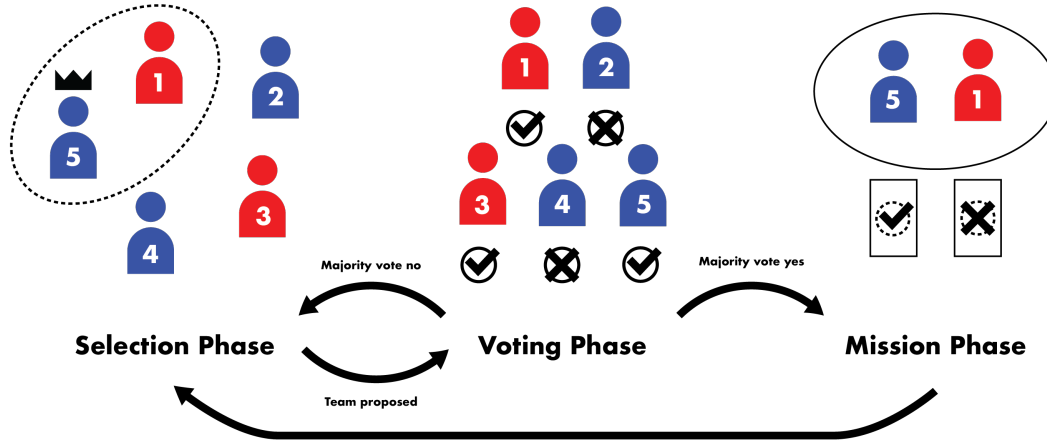


Figure 8: **The three phases per round of Resistance game.** Good players are shown in blue, while Evil players in red. In *Selection Phase*, the team leader (player 5 in this round) proposes a team (player 1 and 5, himself). In *Voting Phase*, all players vote publicly whether to approve this team or not. If the strict majority votes yes, the team is approved and moves on to the mission phase. Otherwise, redo the *Selection Phase* with the next player as leader. If the team goes on the *Mission Phase*, selected team members (player 1 and 5) anonymously vote to pass or fail the mission. If at least one person (player 1, as he is the evil player) votes fail, the mission fails. Otherwise, it succeeds.

We describe the game in more detail here. There are four phases in the game where players need to make decisions: (1) **team selection phase**, (2) **voting phase**, (3) **quest phase**, and (4) **assassination phase**. The game alternates between the first three phases until the end condition is reached, at which point we move on to the assassination phase. Each phase also contains **discussion** where players can challenge others, defend themselves, and negotiate. A flowchart of the game is presented in Figure 10, and an Avalon Rule Prompt is included in Section A.4.

A.1 ROLES

There are four basic roles in Resistance Avalon: **Servant** of Arthur, **Minion** of Mordred, **Merlin**, and **Assassin**. The **Servant** is a basic good character who does not know the identity of any of the other players. The **Minion** is a base evil character who knows who is good and evil but does not know the specific roles of each player. **Merlin** is a unique good character who knows who is good and evil. The **Assassin** is a unique evil character who knows who is good and evil, and in addition, has the ability to assassinate a character at the end of the game. If that character is **Merlin**, the evil team wins.

Good players will always outnumber evil players. Hence, evil players must pretend to be good in order to be voted in on teams (and thus sabotage missions). **SERVANTS** will thus need to sniff out the evil players through their actions and dialogue. **MERLIN** is usually the only good player with additional information, so they will need to discreetly guide the **SERVANTS** in the right direction. **Servants** also need to protect **MERLIN**, so a common strategy is for **SERVANTS** to pretend to have hidden information so that evil players will think that they are **MERLIN**. Evil players will be trying to sniff out **MERLIN** at the same time, so deduction skills are required for all roles.

A.2 ACTIONS FOR EACH PHASE

Depending on the phase **team selection**, **voting**, **quest**, and **assassination**, players may conduct different actions. We detail the specific actions that players can take in each of these phases below.

During the **team selection phase**, only the current *leader* has to make a choice. Leadership passes around the players sequentially in a loop. The action space of **team selection** for the leader consists of all subsets of the players with size equal to the mission team size. The mission team size is different

810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863

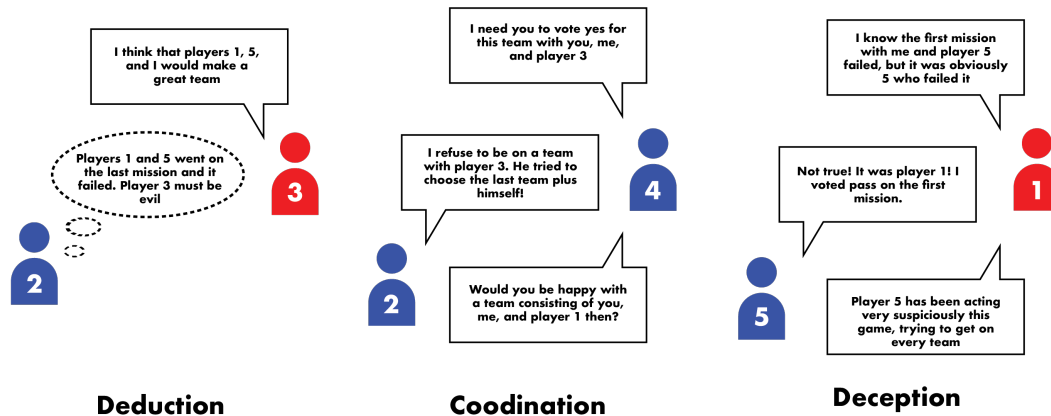


Figure 9: **Communication Skills required to play Avalon.** 1) First, they use logical reasoning to analyze the voting pattern and dialogue of other players and deduce their motives. 2) they must coordinate, communicate, and persuade their teammates to follow a particular strategy. 3) they must also hide their identity and motives through deception.

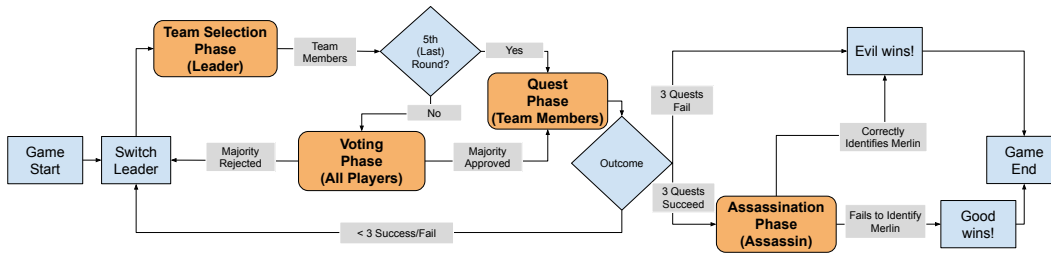


Figure 10: **Flowchart illustrating the various game states and transition diagram.** Round boxes indicate game states (phases) where the player (role highlighted in bracket) has to make decisions

for each mission and is determined by the total number of players in the game. For example, in a 5-player game, on mission No.4, the mission team size is 3, so any subset of {1, 2, 3, 4, 5} with size 3 would be a valid action. After the team proposal is determined by the leader, we move on to the **voting phase** with the selected players.

During the **voting phase**, every player in the game needs to simultaneously vote either APPROVE (1) or REJECT (0). Votes are publicly revealed to all players, so players can see what other players voted. If a strict majority votes APPROVE (1), we then move on to the quest phase with the team that was approved. Otherwise, we move back to the selection phase. Note that if four teams have been rejected in a row, and this is the fifth time a team is proposed (for the same mission), we skip the voting and move directly to the **quest phase**. This prevents the game from dragging on forever.

During the **quest phase**, each selected player on the approved team votes anonymously to either PASS (1) or FAIL (0) the mission. The number of votes of PASS vs FAIL are then revealed to everybody. If the number of FAILs is greater than or equal to the number of FAILs required for the mission to fail (usually 1), then this mission is marked as a failure. Otherwise, this mission is marked as a success. Hence, good players usually have no incentive to fail missions, while evil players will want to have enough failures to pass the failure threshold. If three out of five missions fail, evil wins immediately. Otherwise, if three out of five missions succeed, we move on to the assassination phase.

A.3 DISCUSSION

Group discussion occurs between the **quest** and **selection** phases, as well as right before the **assassination** phase. Players may not communicate during any other time. All conversations are public, and there is no private communication. Typically players may discuss in any format of their choosing as long as only one person is speaking at a time. Some examples of formats include a

864 natural (spontaneous) seminar style (most common, where there is no fixed order of speaking), or
865 sequentially (where players speak in some predefined order). Interruptions and arguments between
866 two players are very common between human players.

867 Usually, players will spend this time discussing a couple of key topics, including (1) the **observations**
868 they made, (2) the *guessed identities and sides* of players, and (3) the **plan** for the next mission.
869 The team leader will usually spend this time asking for advice on what team to select and gathering
870 support for that team. Persuasion and adhering to the preferences of other players are usually key to
871 getting a team approved. Players can also accuse other players of being evil, though arguments will
872 need to be justified in order to be persuasive.

873 For example, a player (player 3) could start off by stating their (1) **observations** of what happened in
874 the previous mission. One FAIL was observed, so at least one player on the previous team (consisting
875 of players (1,2,3)) is evil. Player 3 then emphasizes that both Players 1 and 2 voted APPROVE for
876 the previous mission, which ended up a failure. Moreover, the team was proposed by Player 1 in
877 the first place. Player 3 then moves on to discuss the (2) **identities** of other players. The player
878 says that, despite the fact that only one FAIL was observed, both Players 1 and 2 are evil since they
879 both voted to APPROVE previously. Player 0 is probably good since they voted to REJECT in the
880 previous mission, and Player 3 is also good since they also voted to REJECT, even though they were
881 on the mission. Player 3 then says what they think the (3) **plan** should be. Specifically, Player 3 says
882 that they should reject the current team no matter what since Player 2 is the leader and is evil. The
883 leadership will then pass to Player 3, who will choose the team (0, 3, 4), which good players should
884 vote to approve since it does not contain any suspected evil players¹.

885
886
887
888
889
890
891

892 A.4 GAME ENDING AND ASSASSINATION

893
894
895
896

897 In classic RESISTANCE, a good team wins immediately if three missions are successful. In RESIS-
898 TANCE AVALON, there is an additional assassination phase if three missions are successful. During
899 the **assassination** phase, the ASSASSIN player chooses one player to assassinate. If that player is
900 MERLIN, then evil wins. Otherwise good wins.

901 Before they assassinate a player, the ASSASSIN player can and is encouraged to discuss with the
902 other players (mostly their teammates). good players are also welcome to join in on this discussion
903 to mislead the evil players, though it rarely helps. Players can discuss in a format of their choosing,
904 though there is usually a time limit on how long players can discuss before reaching a decision.

905
906
907
908
909
910
911
912
913
914
915
916

917 ¹At this point, Player 2 reveals that they are the assassin and assassinates Player 3, who is indeed MERLIN.
Player 3's intuition and analysis were way too correct to be a SERVANT

918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971

Avalon rules prompt

The game you are interested in is called The Resistance: Avalon. The Resistance: Avalon is the game of hidden identities and social deduction. There are two teams in the game: Good and Evil. Each player has a hidden identity (role) and side.

There are five Quests in the game and five turns, one for each quest. Good players aim to help three Quests succeed, while Evil players aim to fail three Quests. Different quests require different numbers of players to participate.

At the beginning of the game, each player is assigned a role secretly and randomly. Private information is then revealed to each player. A random player is selected as the leader for the first round.

Each round, after a round of discussion, the leader will select a team of players to participate in the Quest. Then, all players will vote on whether to approve or reject the team publicly. If the team is approved (a strict majority vote to approve), the Quest will be carried out. If the team is not approved, the next player becomes the leader and the next round will start. If four teams are rejected in a row, the fifth team will automatically be approved.

If the team is approved, each team member chooses to pass or fail the Quest anonymously. Usually, if there is at least one failed vote, the Quest fails. Otherwise, the Quest succeeds. In either case, we move on to the next turn and the next quest.

Below are the roles in the game:

Servant of Arthur (Servant): A Good player who does not know who is on the Evil side. The Servant's job is to make sure that the three Quests succeed.

Minion of Mordred (Minion): An Evil player who knows who is on the Evil side. Minion's job is to fail three Quests without being identified by the Good players.

Merlin: A Good player who knows who is on the Evil side. Merlin's job is to make sure that the three Quests succeed without revealing himself to Evil.

Assassin: An Evil player who knows who is on the Evil side. Assassin's job is to assassinate Merlin if the Evil players can identify who Merlin is. If the Assassin successfully assassinates Merlin, the Evil players win the game immediately, even if three quests succeed.

Hence, Evil players usually know who is on the Evil side, but Good players usually do not know who is on the Evil side.

Players may make any claims during the game, at any point in the game. Discussion, deception, accusation, persuasion, and logical deduction are all equally important in order for Good to prevail or Evil to rule the day. Hence, players should rarely reveal their true identity to other players. Players will, can, and should lie to achieve their goals.

972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025

B GAME OF PURE STRATEGY (GOPS) GAME DESCRIPTION

Game of Pure Strategy (GOPS) is a card game for two or more players with a standard deck of card, which is commonly used as an example of multi-stage move game in artificial intelligence (Wikipedia contributors (2023)). In our experiments we play 5 or 6 card GOPS. Specifically, the score cards are $\{1, 2, \dots, n\}$ and each player starts with a hand of cards $\{1, 2, \dots, n\}$ where n is the number of cards and rounds. The GOPS rules prompt is included in this section below.

GOPS rules prompt

The game you want to write a function for is GOPS (game of pure strategy), also known as Goofspiel. The game has two players, and is played with a deck of score cards. Each player is dealt the same hand of cards at the beginning. The goal of the game is to get a score higher than your opponent. At the beginning of each round, a score card is randomly drawn without replacement from the score deck. Then each player plays a card simultaneously from their hand. The player who plays the higher card wins the round and gets the score card. They add the score of the score card to their total score. If the two cards played are the same, the person who wins the next round will get both score cards. The game continues until all score cards have been played. The player with the highest total score wins the game.

1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079

C LIMITATIONS

While our method performs better on average, individual runs can have high variance. Since the performance of an agent in multi-agent adversarial game settings is highly dependent on opponents' policies, feedback from these environments tend to be highly noisy, with noise increasing with the number of players. This is especially true when learning Avalon heuristics, where the performance depends on the policies of 5 other players, teammates and opponents. We believe that running more game simulations with different opponent policies can help reduce this feedback noise. We also acknowledge the inherent noisiness in LLM generations and how that can impact our results. We tried to reduce this noise by (1) using the same seed functions when benchmarking the different LLM improvement methods and (2) collecting generated strategies from multiple runs. We also did not test our method on other non-adversarial environments such as question answering and text-based worlds. However, given the strong performance of our method in adversarial multi-agent settings, we believe that similar performance will be observed in single agent, non-adversarial settings.

1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133

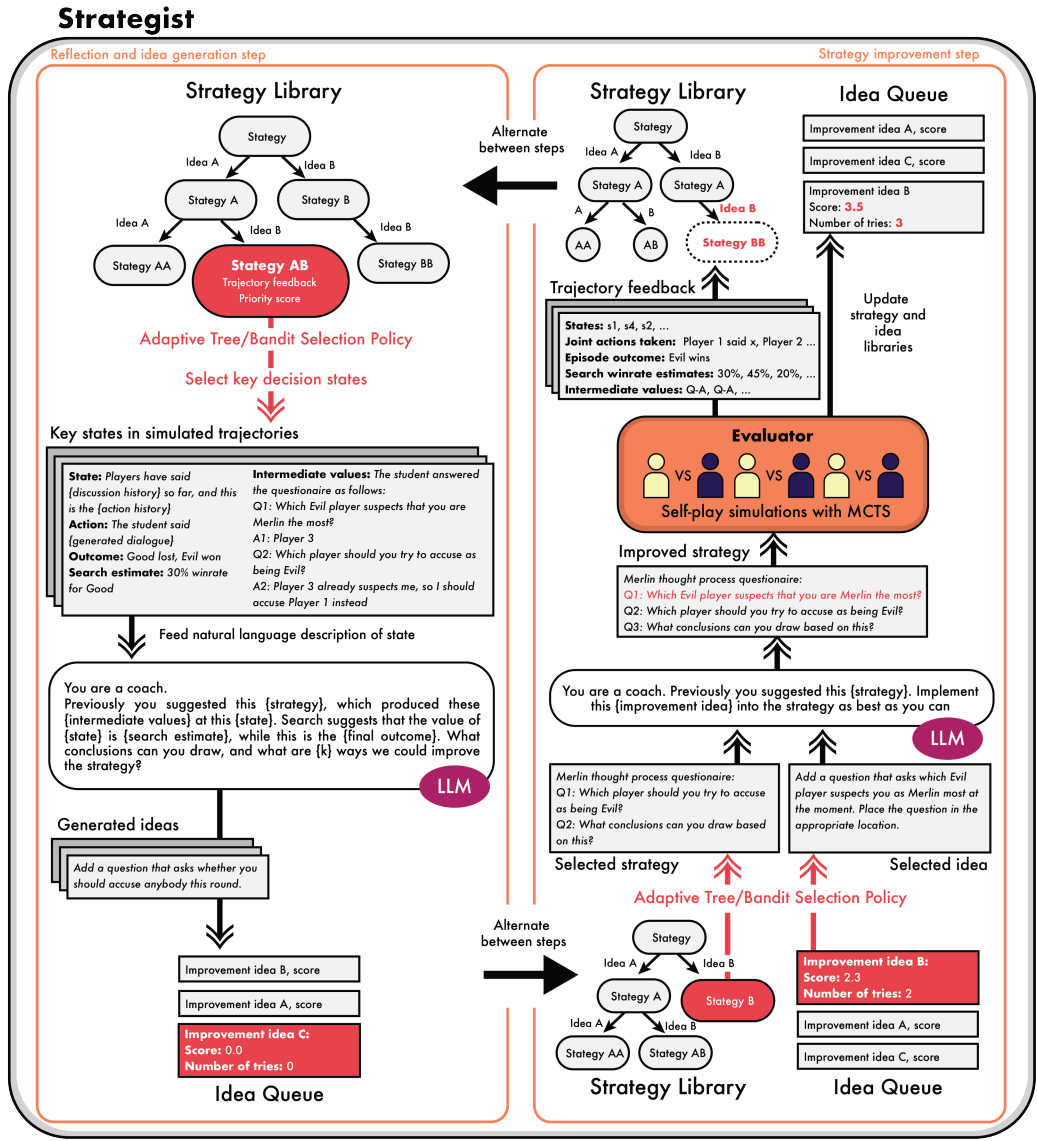


Figure 11: Overview of our high-level strategy improvement method: The process alternates between two steps—idea generation and strategy improvement.

D IMPROVEMENT PROCESS IMPLEMENTATION DETAILS

Our method alternates between two main steps: **idea generation** and **strategy implementation**, as illustrated in Figure 11. These steps are structured around the optimization of strategies stored in a hierarchical tree structure T . Below, we detail the key components of our implementation.

D.1 STRATEGY TREE AND IDEA QUEUE

The **strategy tree** T maintains all discovered strategies, their associated feedback (τ_s), and priority scores (z_s), which determine their likelihood of being selected for further refinement. Complementing this is the **idea queue** Q , a priority-based queue that stores candidate ideas generated during the improvement process. Each idea $d \in Q$ is initialized with a prior score $z_d = 0.0$ and a usage count $n_d = 0$.

```

1134 Algorithm 1: STRATEGIST Pseudocode
1135
1136 Data:  $T$ : strategy tree storing strategy  $s$ , feedback ( $\tau_s$ ), and priority score ( $z_s$ ),  $Q$ : idea queue, ‘seed
1137 functions’,  $N_{ideas}$ : number of ideas,  $N_{strategies}$ : number of strategies,  $N_{evolutions}$ : number of
1138 evolutions,  $N_{feedback\_examples}$ : number of states of give as feedback,
1139 Function select_strategy( $T$ ):
1140    $\sigma_{best} \leftarrow \arg \operatorname{softmax}_{\sigma \in T_2} z_\sigma$  // one possible implementation where you take one
1141   of the best two strategies in the whole tree randomly (BFS2)
1142   return  $S_{best}$ 
1143 Function select_idea( $Q, \sigma$ ):
1144    $d_{best} \leftarrow \operatorname{softmax}_{d \in Q} UCB(z_d, n_d)$  // one possible implementation where you
1145   take the best strategy in the queue using softmax UCB,  $z_d$  being
1146   the empirical  $q$ -value and  $n_d$  being the number of tries
1147   return  $\sigma_{best}$ 
1148 Function select_key_states( $\tau_\sigma$ ):
1149    $K_\sigma \leftarrow \arg \max_{s \in \tau} (\operatorname{SearchEstimate}(s) - v_\sigma(s))^2$  // one possible way to select key
1150   states for  $\sigma$  that is a value heuristic  $v_\sigma$ 
1151   return  $K_\sigma$ 
1152 Function generate_ideas( $N_{ideas}$ ):
1153    $\sigma \leftarrow \operatorname{select\_strategy}(T)$ ;
1154    $K_\sigma \leftarrow \operatorname{select\_key\_states}(\tau_\sigma)$  //  $K_\sigma$  is a set of key states from the
1155   trajectory feedback  $\tau_\sigma$  for strategy  $\sigma$ 
1156    $D_{new\ ideas} \leftarrow \operatorname{LLM}(\operatorname{Generate } N_{ideas} \text{ new ideas based on string description of } K_\sigma, \text{ which includes the}$ 
1157    $\text{output of the strategy, action taken, state description, final outcome of the trajectory, search estimate of}$ 
1158    $\text{the state, and any intermediate values used to compute the output of the strategy});$ 
1159   for  $d \in D_{new\ ideas}$  do
1160     Store  $d$  in  $Q$  with prior score  $z_d = 0.0$  and  $n_d = 0$ ;
1161   end
1162 Function implement_strategies( $N_{strategies}$ ):
1163    $\Sigma_{new}, D, P = [], \{\}, \{\}$  // list of new generated strategies, dictionary
1164   mapping new generated strategy to the idea that generated it,
1165   and dictionary mapping generated strategies to their parents
1166   for  $i \leftarrow 1$  to  $N_{strategies}$  do
1167      $\sigma \leftarrow \operatorname{select\_strategy}(T)$ ;
1168      $d \leftarrow \operatorname{select\_idea}(Q, \sigma)$ ;
1169      $\sigma_{new} \leftarrow \operatorname{LLM}(\operatorname{Improve } \sigma \text{ using } d)$ ;
1170      $\Sigma_{new}.append(\sigma_{new})$ ;
1171      $D[\sigma_{new}] = d$ ;
1172      $P[\sigma_{new}] = \sigma$ ;
1173   end
1174    $W, \mathcal{T} \leftarrow \operatorname{SelfplaySimulate}(\Sigma_{new} \cup \operatorname{unique}(P.values))$  // simulate games, getting
1175   average winrates  $W[\sigma]$  for each strategy  $\sigma$  and simulated
1176   trajectory feedback  $\mathcal{T}[\sigma]$ 
1177   for  $\sigma \in \Sigma_{new}$  do
1178      $T.add(\sigma, P[\sigma], D[\sigma])$  // add new strategy to tree from parent based on
1179     idea
1180      $z_\sigma \leftarrow W[\sigma]$  // add function score
1181      $z_{D[\sigma]} \leftarrow \frac{n_{D[\sigma]}}{n_{D[\sigma]}+1} z_{D[\sigma]} + \frac{1}{n_{D[\sigma]}+1} (W[\sigma] - W[P[\sigma]])$  // update idea score with
1182     how much it improved the strategy by
1183   end
1184 repeat
1185   generate_ideas( $N_{ideas}$ );
1186   implement_strategies( $N_{strategies}$ );
1187 until  $N_{evolutions}$ ;
1188 return Best strategies in  $T$  according to their scores  $z_s$ 

```

D.2 KEY FUNCTIONS

Selecting Strategies (SelectStrategy) To identify promising strategies for refinement, we use a softmax over the priority scores z_s stored in T . One implementation chooses between the two highest-priority strategies in a breadth-first search order.

1188 **Selecting Ideas (SelectIdea)** Ideas are selected from Q using an Upper Confidence Bound
1189 (UCB) approach. Each idea’s selection balances its empirical improvement potential (z_d) with its
1190 exploration term n_d , ensuring both promising and underexplored ideas are considered.

1191
1192 **Selecting Key States (SelectKeyStates)** Key states K_σ for a given strategy σ are selected
1193 based on a heuristic that emphasizes discrepancies between a search-based estimate and the strategy’s
1194 value function v_σ . These states highlight areas where the strategy can be improved.

1195

1196 D.3 IDEA GENERATION

1197
1198 During the **idea generation step**, the algorithm:

- 1199 1. Selects a strategy σ from T using `SelectStrategy`.
- 1200 2. Extracts key states K_σ from the trajectory feedback τ_σ using `SelectKeyStates`.
- 1201 3. Uses a language model (LLM) to propose N_{ideas} new ideas based on the string description
1202 of K_σ . This description includes the strategy’s actions, state details, trajectory outcomes,
1203 search estimates, and intermediate values.
- 1204 4. Adds the generated ideas to Q with default priority and usage counts.

1205

1207 D.4 STRATEGY IMPLEMENTATION

1208
1209 In the **strategy implementation step**, the algorithm:

- 1210 1. Selects a strategy σ from T and an idea d from Q using `SelectStrategy` and
1211 `SelectIdea`.
- 1212 2. Refines σ using d , producing a new strategy σ_{new} .
- 1213 3. Simulates self-play games for σ_{new} and related strategies to evaluate performance (win rates
1214 $W[\sigma]$) and collect trajectory feedback ($\mathcal{T}[\sigma]$).
- 1215 4. Updates the strategy tree T with σ_{new} , linking it to its parent strategy and the idea that
1216 inspired it.
- 1217 5. Updates the priority scores z_σ for strategies and z_d for ideas based on the observed improve-
1218 ments.

1219

1221 D.5 ITERATIVE EVOLUTION

1222
1223 The improvement process repeats for $N_{\text{evolutions}}$ iterations. In each iteration, new ideas and strategies
1224 are generated and evaluated, incrementally building a tree of optimized strategies.

1225

1226 D.6 SCALABILITY AND EFFICIENCY

1227
1228 Our method leverages efficient UCB-based selection and hierarchical feedback propagation to scale to
1229 large T and Q . By iteratively refining strategies through targeted improvements, it ensures continuous
1230 enhancement while maintaining computational feasibility.

1231

1232

1233

1234

1235

1236

1237

1238

1239

1240

1241

1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295

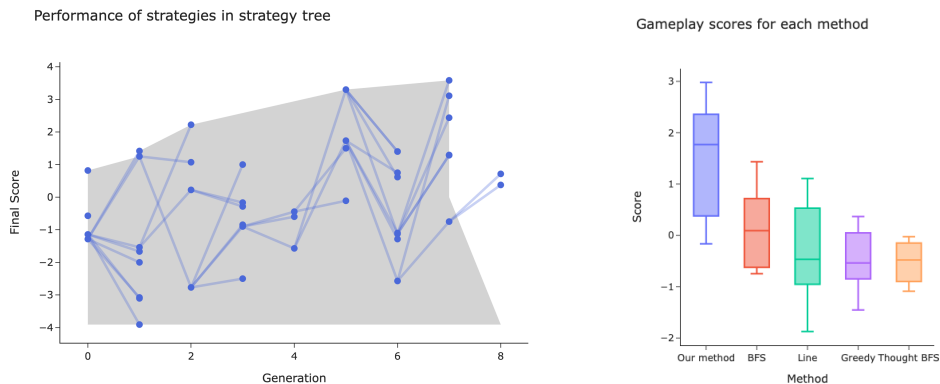


Figure 12: **Left:** Example performance of value heuristics strategy tree for GOPS. Points indicate the final evaluation scores and the generation of improved functions, with lines showing the evolutionary path between functions. Our method successfully escapes local maxima and continues exploring the strategy space. These scores reflect final gameplay results against a fixed set of opponents, differing from the intermediate self-play scores used to guide strategy improvements. **Right:** Comparison of different improvement methods on 6-card GOPS, where 9 functions generated by each method play against one another over 1024 total games.

E AVALON AGENT IMPLEMENTATION DETAILS

We describe in detail how we implement our model below and as shown in figure 13. Unless otherwise specified, the word ‘action’ will refer to non-dialogue actions. Note that we do not conduct search over raw dialogue space since that is not very computationally feasible. Instead, we search over intended actions and condition our dialogue on that.

Specifically, the language component consists of a dialogue analyzer and a dialogue generator, while the moves component consist of the action planner. Whenever the agent needs to speak, they first analyze what was said so far in the current discussion round using the dialogue analyzer. The dialogue analyzer, with the help of an LLM, updates the internal beliefs of the agent. For example, in Avalon, internal beliefs might include the probability that the agent assigns to each other player of being Evil and of being Merlin. These beliefs are then passed to the action planner, which uses them to figure out the best next move, i.e. the action intent. The action intent is then passed to the dialogue generator, which generates dialogue with the help of an LLM. When the agent needs to take a move, we run through the same process except that the agent takes the action intent as the move and no dialogue is generated.

E.1 DIALOGUE ANALYZER (DISCRIMINATOR)

The dialogue analyzer f_{ana} takes as input I information set (partial information) of the current state for the player, d_t the discussion so far this round, and b some prior beliefs about the hidden state of the game, and returns \hat{b} , the updated beliefs, and $\hat{\Pi}_t$, the predicted joint action policy of the all the players (i.e. the action intent) for the next action step t . Recall that simultaneous games can be expanded as partial information games, where the simultaneous moves are treated as hidden information. Hence, we are essentially predicting a distribution over the hidden states s given the information set I using the dialogue analyzer.

$$\hat{b}, \hat{\Pi}_t = f_{ana}(I, d_t, b)$$

In the context of Avalon, I will contain information such as (1) the dialogue this round so far (2) summary of the dialogue from previous rounds (3) mission track record (4) historical record of actions taken by players in previous rounds, and (5) private information of the player such as who is Good and Evil. b will contain information on (1) the probability of each player being Evil and (2)

1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349

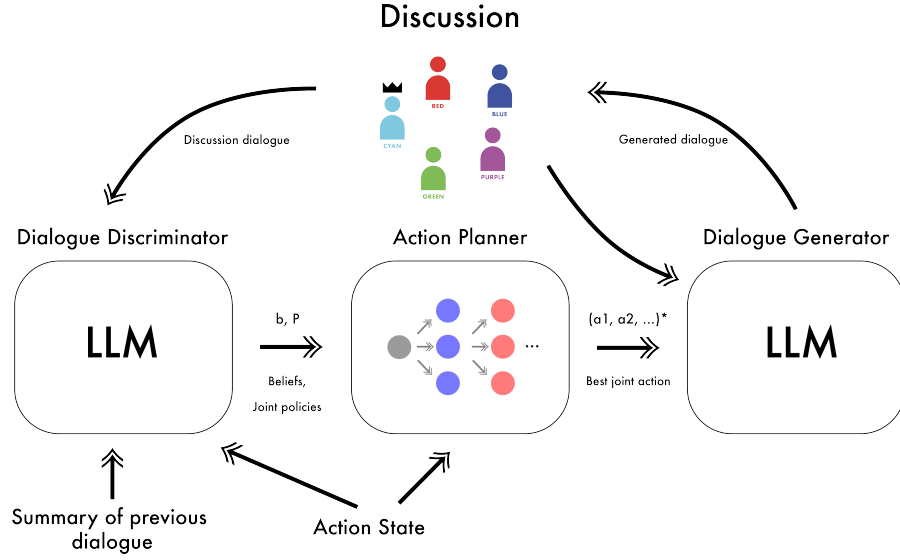


Figure 13: Overview of the LLM-powered agent, including the three main modules that we use to generate dialogue during discussion

the probability of each player being Merlin, both conditioned on the private information contained in \mathbf{I} . While a full treatment of the distribution over the hidden state space \mathcal{S} we require assigning probabilities to each possible combination of Good and Evil players, not just assessing the marginal probability of each player being Good individually, in practice

We implement f_{ana} using an LLM, which is fed \mathbf{I} , \mathbf{d} , \mathbf{b} (converted to natural language form) as prompts, along with some instruction prompt ϕ_{ana} that prompts it to produce $\hat{\mathbf{b}}$, $\hat{\Pi}_t$. Specifically,

$$f_{ana}(\mathbf{I}, \mathbf{d}_t, \mathbf{b}) = f_{LLM}(\phi_{dis}, \mathbf{I}, \mathbf{d}, \mathbf{b})$$

We show examples of such prompts in Appendix J.

E.2 ACTION PLANNER

Given $\hat{\mathbf{b}}$ the belief prior, $\hat{\Pi}_t$ the predicted joint action policy for all players, and s the representation of the current state, the action generation model f_{act} generates a probability distribution over possible actions π^i for the main player i that is the best response to $\hat{\Pi}_t$. We do so by using search techniques to look ahead and find the best response.

$$\pi^i = f_{act}(\hat{\mathbf{b}}, \hat{\Pi}_t, \mathbf{I})$$

More specifically, in our search implementation, at the first layer, we first sample across possible hidden states $s \sim \hat{\mathbf{b}}$ according to the belief prior. At the second layer (i.e. the first action stage t), we calculate expected q -values for each action $a \in \mathcal{A}$ that the main player can take if the other players play actions $\mathbf{a} \sim \hat{\Pi}_t$ according to the predicted joint distribution. In subsequent action stages, the search process will assume that other players play according to their policy simulated and induced by the value heuristic that is not dialogue dependent. We then take the best response action $a_i^* = \max(\pi^i)$ as the intended action. Since this is a partial information game, expected q -values are taken across information sets, not states. We describe how our action planner is implemented in more detail in Appendix F.

1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403

E.3 DIALOGUE GENERATION

The dialogue generator f_{gen} takes as input \mathbf{I} some representation of the current information set and a_i^* , the intended best response action, and outputs dialogue d .

$$d = f_{gen}(\mathbf{I}, a_i^*)$$

We will implement f_{gen} using an LLM, which is fed \mathbf{I} and a_i^* directly as prompts, along with some instruction prompt ϕ_{gen} that prompts it to produce realistic sounding dialogue that helps it achieve its intended action.

For example, perhaps the player wants to approve the next team. Then it should try to generated dialogue that convinces the other players to also approve.

We show examples of such prompts in Appendix J.

F VALUE HEURISTIC IMPLEMENTATION DETAILS

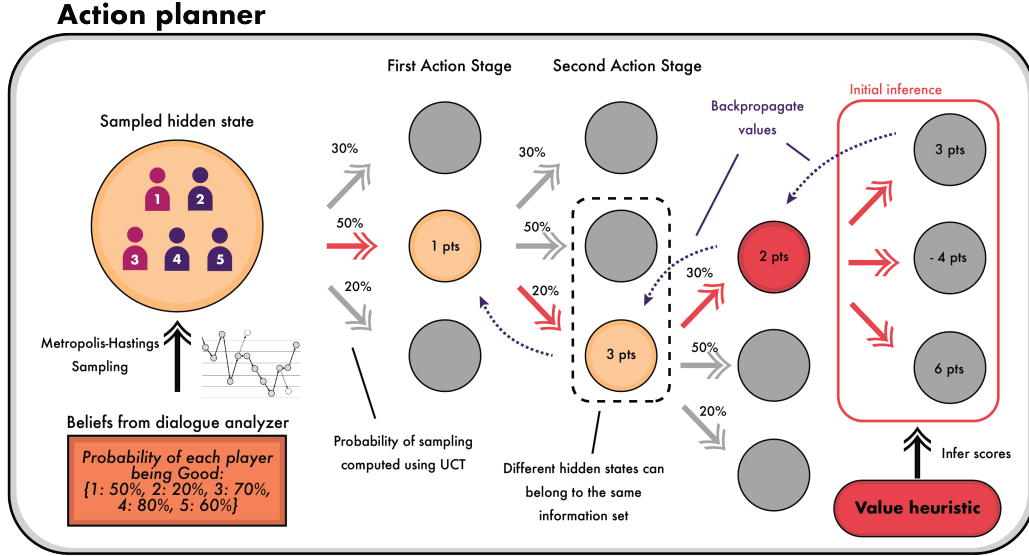


Figure 14: **Overview of how we utilize the trained value heuristics in MCTS tree search to get a non-dialogue based policy.** While we only display the values for a single player in the diagram, note that in practice we infer and update the values for all players at the same time. Next states are sampled using the PUCT formula we described. We sample an initial hidden state based on the internal beliefs of the agent. Metropolis-Hastings is used to sample since it may be difficult to calculate the probability density specified by the internal beliefs. Note that values estimated using MCTS are also passed as feedback to the evaluator.

The MCTS search process is depicted in Figure 14, where we simulate a trajectory from the hidden state we are at until we reach some unexpanded state s . The probability of transitioning to a state during simulations is computed assuming that each player samples from their optimal actions according to their PUCT (polynomial upper confidence trees) values (and ϕ_e for the environment actor) (Schrittwieser et al., 2020). Since in some environments players may only be able to observe information sets, when computing the PUCT values we average over all expanded states in that information set. Moreover, the initial hidden state can be sampled according to a prior (or empirical prior) over the states in the information set that the player observed. Then, using our value heuristic, we compute the values of each of the next hidden states. We then backpropagate our new values back up the simulated trajectory, updating the intermediate states. After running a few MCTS simulations (roll-outs) like the one we described, the planner then outputs the action which leads to the highest value next state. We show our information set PUCT formula below, where $N(s, a)$ is the number of times we took action a at state s during MCTS rollouts, $P(s, a)$ is the prior probability of selecting action a from state s , C is the exploration constant, Q_{emp} is the empirical average of MCTS roll-out outcomes, $\hat{Q}(s, a)$ is the prior computed by our value heuristic, α controls how much weight be put on the prior (often $\alpha = 1$), and π_B is the distribution across hidden states in the information set given our **beliefs** B , some parametrization of π_B . Since π_B is often hard to compute, we can simply set $\pi_B(s|I) = \frac{\sum_b N(s,b)}{\sum_{s' \in I} \sum_b N(s',b)}$ to be the empirical roll-out distribution, given that we sample initial states $s_0 \sim \pi_B(s_0|I)$ according to our beliefs. For example, in Avalon, we can sample the hidden roles according to our beliefs B using Metropolis-Hastings for the initial state s_0 .

$$Q(s, a) = \frac{N(s, a) \cdot Q_{emp}(s, a) + \alpha \cdot \hat{Q}(s, a)}{N(s, a) + \alpha} \quad (1)$$

$$\text{PUCT}(I, a) = \sum_{s \in I} \pi_B(s|I) \left[Q(s, a) + C \cdot P(s, a) \cdot \frac{\sqrt{\sum_b N(s, b)}}{1 + N(s, a)} \right] \quad (2)$$

G DIALOGUE GUIDE IMPROVEMENT EVALUATION IMPLEMENTATION DETAILS

We provide more details on our dialogue improvement evaluation process here and as shown in figure 15. The improvement method (skill coach) remains the same as we described before.

We first generate a synthetic dataset by simulating a game of Avalon with initial dialogue and move policies ϕ . Given the dialogue guide σ we want to evaluate, we then sample ‘scenarios’ from the dataset. A scenario consists of a game state, intended action, and private information in the simulated trajectory. We create an Avalon agent like the one we described in E for each player in the game, initialized with their corresponding private information. The Avalon agent is then asked to generate dialogue using the dialogue guide σ .

Using this new generated dialogue, we then simulate the next round of dialogue analysis for each Avalon agent. This produces analysis scores based on how likely they think the player is to be Merlin z_{merlin} , and how likely they think the player is to be Evil z_{evil} , where $z_{merlin}, z_{evil} \in [-2, 2]$. For evaluating Merlin, we get the average z_{merlin} scores from the Evil players, \bar{z}_{merlin} , along with the average z_{evil} scores from the Good players \bar{z}_{evil} . We then take the minimum of these two as the feedback score $z = \min\{\bar{z}_{evil}, \bar{z}_{merlin}\}$. This is because Merlin wants to both minimize the probability of being detected by the Evil players, and also minimize the probability of being identified as Evil by the Good players.

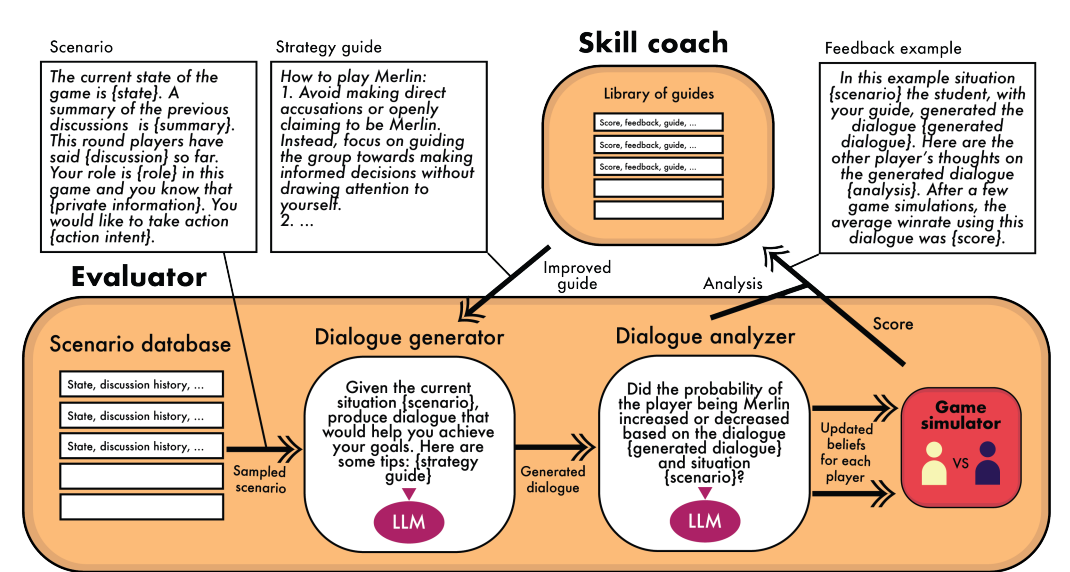


Figure 15: Overview of our improvement process for learning dialogue generation strategies. This includes how we evaluate the dialogue and how we collect feedback. The skill coach here can be implemented as either our improvement method, STRATEGIST, or any of the baseline methods we described.

The dialogue analyzer (discriminator) is described in more detail in Appendix E and the specific generation and analysis prompts are shown in Appendix J.

1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565

H VALUE HEURISTIC LLM PROMPT AND OUTPUT EXAMPLES

H.1 SYSTEM PROMPTS

System prompts are guidelines for LLM to generate outputs align with the intended goals. In our case, the goal is to generate a function that evaluates the value of a state in a game under low cost.

Value heuristic system prompt

You are a function engineer trying to write a function that can evaluate the value of a state in a game. This is known as a value heuristic, and will be used in look-ahead search algorithms to evaluate the value of unexplored states. Your goal is to develop a heuristic that is as accurate as possible without being too expensive to compute. Hence, you are not allowed to run simulations in the function.

The following example is a detailed prompt telling the LLM how to format the value heuristics specifically in the GOPS game. The format of input and output are clearly defined in the prompt with illustrations, examples and structures.

1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619

GOPS value heuristics function signature

The function (written in python) should be named 'evaluate state' and take in a tuple called 'state' of the game state as input. Specifically, the input tuple will be of length 9, and it should return 2 elements. The first element should be a tuple with 2 floats: the first element being the score you expect player 0 will get at the end of the game, and the second element being the score you expect player 1 will get at the end of the game. The second element should be a dictionary of any important intermediate values that you used to calculate the scores. For example, if you think player 0 will win 12 total points by the end of the game and player 1 will win 8 total points, the function should return (12, 8).

Make sure your output only includes the code of the function itself in plain text such that it is executable using exec() in python. Any helper functions should be defined within the scope of the function 'evaluate state'. Include comments in your code so that it is readable, but everything should be implemented.

The signature of the function should be as follows:

```
def evaluate_state(state) -> tuple[tuple[float, float], dict]:
    score_cards = state[0] # a python list of the score cards (integers) that have been played, in the
    order they were played
    player_0_played_cards = state[1] # a python list of the cards (integers) player 0 has played, in the
    order they were played.
    player_1_played_cards = state[2] # a python list of the cards (integers) player 1 has played, in the
    order they were played.
    is_turn = state[3] # bool, true if it is you and your opponent's turn to play, false if it is time
    to draw a new score card
    player_0_score = state[4] # float or integer, player 0's score so far
    player_1_score = state[5] # float or integer, player 1's score so far
    score_deck = state[6] # a python set of the score cards (integers) left in the deck, either same
    length as player_0_hand and player_1_hand or one less since the score card appears before the
    players play. May be empty
    player_0_hand = state[7] # a python set of the cards (integers) left in player 0's hand. May be
    empty
    player_1_hand = state[8] # a python set of the cards (integers) left in player 1's hand. May be
    empty
    # explanation of what we do next
    ...
    <intermediate_value> = value1
    # explanation of what we do next
    ...
    <intermediate_value> = value2
    # explanation of what we do next
    ...
    player_scores = (player_0_expected_score, player_1_expected_score)
    intermediate_values = {'<intermediate_value>': intermediate_value1, '<intermediate_value>':
    intermediate_value2, ...}
    return player_scores, intermediate_values # make sure the return is exactly in this format
```

Where you can use your own names for the intermediate values and the values themselves. Please start with "def evaluate state(state):"

H.2 IDEA GENERATION EXAMPLES

The idea generation prompt included system prompt, game rules, previous guide and feedback reflections. Following those four components, we construct the format and an example of ideas to guide the generation of LLM.

1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673

Prompt for idea generation

<System prompt>
<Game rules>
<Previous guide>
<Feedback reflections>

Based on the function, feedback, and conclusions you drew, what are 2 improvements that you can make to the function that you think will have the most impact? Be as specific and concrete as possible, and write them out in the following format:

- Thoughts: <your thoughts here>
- Idea 1: <your idea here>
- Idea 2: <your idea here>

...

Here's an example of what this might look like for 3 improvement ideas:

- Thoughts: I should consider the number of cards left in the deck when evaluating the value of a state.
- Idea 1: I should add a term to the value function that penalizes states where there are fewer cards left in the deck.
- Idea 2: I should add a term to the value function that rewards states where the player has more cards in their hand than the opponent.
- Idea 3: I should add a term to the value function that rewards states where the player has more cards in their hand than the opponent and there are fewer cards left in the deck.

Below is an instance of Feedback of GOPS game, showing the setup of two players, the intermediate values involved in the computation, and the actual scores.

1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727

Feedback example

Example 9:

The state you were trying to estimate a value for is:

The current state of the game is as follows:

- The score cards that have been revealed are: (2, 4, 5, 1, 3)
- The cards that player 0 has played are: (1, 2, 4, 3, 5)
- The cards that player 1 has played are: (3, 5, 1, 2, 4)
- Player 0's score so far is: 9
- Player 1's score so far is: 6
- The score cards left in the deck are: set()
- The cards left in player 0's hand are: set()
- The cards left in player 1's hand are: set()

The function you generated returned the following values:

```
{0: 3, 1: -3}
```

for the expected end of game scores of the players.

Some intermediate values that you used to calculate the scores were:

```
{'player_0_expected_score': 9, 'player_1_expected_score': 6, 'dynamic_penalty': 0.0, 'player_0_hand_reward': 0, 'player_1_hand_reward': 0, 'player_0_adjustment': 0, 'player_1_adjustment': 0, 'player_0_strategic_adjustment': 0, 'player_1_strategic_adjustment': 0}
```

The estimated end of game scores of the players using lookahead search with your function was:

```
{0: 0.0, 1: 0.0}
```

The actual scores of the players at the end of the game in the simulation were:

```
{0: 3.0, 1: -3.0}
```

Recall that feedback reflection refers to feedback for strategy, which is used to generate ideas. For prompting, We specify that game rules, previous guide and feedback examples are generated previously. The following is the prompt for feedback reflections.

1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781

Prompt for feedback reflections

<System prompt>
<Game rules>
Previously you generated the following function to evaluate the value of a state in the game.
<Previous guide>
Below is some feedback on how the function you generated performed when we tested it. Note that simulations involve high variance and the actual scores may not match the expected scores exactly. Hence, you should focus on trying to get the scores produced by your function to match those predicted by look-ahead search as closely as possible.
<Feedback examples>
Based on the feedback given and the function you generated previously, what are some conclusions you can draw from the feedback? Make sure to cite the specific examples in the feedback to justify your analysis.

Below is the feedback reflections from the LLM.

Feedback reflections

From the feedback provided, we can draw several conclusions regarding the performance of the function in evaluating the value of a state in the GOPS game:

- 1. Discrepancy in Expected Scores:** In several examples (e.g., Example 4, Example 7, Example 8), the expected end-of-game scores returned by the function do not match the scores predicted by look-ahead search closely. This indicates that the function may not be accurately estimating the value of the states in the game.
- 2. Inconsistencies in Strategic Adjustments:** The function includes a strategic adjustment based on the opponent's played cards, but in all cases, the strategic adjustments are zero. This suggests that the strategic adjustments made in the function may not be capturing the true strategic implications of the opponent's moves effectively.
- 3. Dynamic Penalty Adjustment:** The function includes a dynamic penalty adjustment based on the remaining high-value score cards in the deck. However, the dynamic penalty adjustments may not be accurately reflecting the impact of high-value score cards left in the deck on the game outcome.
- 4. Limited Hand Size Reward:** The function rewards states where a player has more cards in their hand than the opponent. While this reward is included, it might not be capturing the full strategic implications of having more cards in hand effectively.
- 5. Simulation vs. Expected Scores:** The actual scores at the end of the game in the simulations do not match the expected scores returned by the function precisely. This indicates that the function's evaluation may not be aligning well with the actual game outcomes.

In conclusion, the function may need further refinement in terms of capturing strategic adjustments, dynamic penalties, and hand size rewards to improve the accuracy of the state evaluation in the GOPS game. Additionally, considering the discrepancies between the expected and actual scores, further testing and adjustments may be necessary to enhance the function's performance in evaluating the value of states accurately.

Also, with the prompts of idea generation, below is the generated new ideas by LLM.

1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835

Generated new ideas

Thoughts: The function needs to better capture the strategic implications of the opponent's moves and adjust the expected scores accordingly. Additionally, the dynamic penalty adjustment and hand size rewards could be improved to better reflect their impact on the game outcome.

Idea 1: Enhance the strategic adjustment component of the function by analyzing the opponent's played cards more deeply. This could involve considering patterns in the opponent's moves, potential card combinations, and predicting future moves based on past actions.

Idea 2: Revise the dynamic penalty adjustment to more accurately reflect the impact of high-value score cards left in the deck. This could involve adjusting the penalty dynamically based on the remaining high-value cards and their likelihood of being drawn in future rounds. This adjustment could help in better assessing the risk associated with certain states in the game.

H.3 STRATEGY IMPLEMENTATION EXAMPLES

During the strategy implementation step, we first select a strategy and an idea from the libraries using the adaptive selection policy. Then, we prompt the LLM to implement the idea on the strategy, generating a new improved strategy.

In this section, we showcase two examples. Each example illustrates strategies before and after enhancements made by the LLM. This comparison highlights the effectiveness of our strategy implementation. The improved parts by LLM are highlighted in yellow.

Below is the prompt for strategy implementation.

Prompt for implementation

```
<System prompt>  
<Game rules>  
Previously you generated the following function to evaluate the value of a state in the game:  
<Previous guide>  
Here is a possible way to improve this function:  
<Improvement ideas>
```

H.3.1 EXAMPLE 1

Below is the GOPS Value Heuristics Function before the strategy improvement.

1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889

```
GOPS Value Heuristics Function (Before)

def evaluate_state(state) -> tuple[tuple[float, float], dict]:
    score_cards = state[0]
    player_0_played_cards = state[1]
    player_1_played_cards = state[2]
    is_turn = state[3]
    player_0_score = state[4]
    player_1_score = state[5]
    score_deck = state[6]
    player_0_hand = state[7]
    player_1_hand = state[8]

    # Calculate the potential scores for each player based on the remaining score cards
    player_potential_score = player_0_score
    opponent_potential_score = player_1_score

    if len(score_deck) > 0:
        remaining_score_cards = len(score_deck)
        if len(player_0_played_cards) > len(player_1_played_cards):
            player_potential_score += remaining_score_cards / 2
        elif len(player_1_played_cards) > len(player_0_played_cards):
            opponent_potential_score += remaining_score_cards / 2

    player_advantage = player_potential_score - opponent_potential_score

    return (player_potential_score, opponent_potential_score), {'player_advantage': player_advantage}
```

Below is the selected idea to improve the above strategy.

Selected GOPS Value Heuristics Improvement Idea

Update the function to calculate the advantage of winning future rounds based on the potential scores each player could achieve with the remaining score cards and their current scores. This would involve considering the number of cards left in the deck, the players' hands, and the current scores to determine the advantage one player may have over the other in the upcoming rounds.

Below is the GOPS Value Heuristics Function after enhancement by LLM. The improved parts by LLM are highlighted in yellow.

1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943

```

GPS Value Heuristics Function (After)

def evaluate_state(state) -> tuple[tuple[float, float], dict]:
    score_cards = state[0]
    player_0_played_cards = state[1]
    player_1_played_cards = state[2]
    is_turn = state[3]
    player_0_score = state[4]
    player_1_score = state[5]
    score_deck = state[6]
    player_0_hand = state[7]
    player_1_hand = state[8]

    # Calculate the potential scores for each player based on the remaining score cards
    # and current scores
    player_potential_score = player_0_score + sum(score_deck) / 2
    opponent_potential_score = player_1_score + sum(score_deck) / 2

    # Determine the advantage of winning future rounds based on potential score
    if is_turn:
        player_potential_scores = [player_potential_score + card for card in player_0_hand]
        opponent_potential_scores = [opponent_potential_score + card for card in player_1_hand]

        player_advantage = sum(player_potential_scores) / len(player_potential_scores)
        - sum(opponent_potential_scores) / len(opponent_potential_scores)
    else:
        player_advantage = player_potential_score - opponent_potential_score

    return (player_potential_score, opponent_potential_score), {'player_advantage': player_advantage}

```

H.3.2 EXAMPLE 2

Below is the Avalon Value Heuristics Function before improvements.

1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997

```
Avalon Value Heuristics Function (Before)

def evaluate_state(state):
    num_successful quests = sum(historical_quest_results)
    num_failed_quests = len(historical_quest_results) - num_successful_quests
    num_remaining_quests = len(num_participants_per_quest) - len(historical_quest_results)

    num_evil = len(players) - num_good
    num_evil_in_quest_team = len([player for player in quest_team if not is_good[player]])

    success_probability = 0.5
    if phase == 0:
        if num_successful_quests >= 3:
            success_probability = 0.9
        elif num_failed_quests >= 3:
            success_probability = 0.1
    elif phase == 1:
        success_probability = 0.8 if num_evil_in_quest_team == 0 else 0.2
    elif phase == 2:
        success_probability = 0.9 if num_successful_quests > num_failed_quests else 0.1
    elif phase == 3:
        if 'Merlin' in roles and 'Assassin' in roles:
            merlin_index = roles.index('Merlin')
            assassin_index = roles.index('Assassin')
            if assassin_index in quest_team:
                success_probability = 0.1
            else:
                success_probability = 0.9

    expected_winrates_per_player = dict()
    for player in players:
        if is_good[player]:
            expected_winrates_per_player[player] = success_probability
        else:
            expected_winrates_per_player[player] = 1 - success_probability

    intermediate_values = {
        'num_successful_quests': num_successful_quests,
        'num_failed_quests': num_failed_quests,
        'num_remaining_quests': num_remaining_quests,
        'num_evil_in_quest_team': num_evil_in_quest_team
    }

    return expected_winrates_per_player, intermediate_values
```

Below is the improvement idea pf Avalon Value Heuristics Function.

```
Avalon Value Heuristics Function Improvement Idea

Incorporate a more nuanced success probability calculation based on the roles of the players and the current phase of the game. For example, in phase 1 (team selection), consider the balance of Good and Evil players on the quest team and adjust the success probability accordingly. This can help better reflect the strategic considerations during team selection.
```

The following is Avalon Value Heuristics Function after enhancements, highlighted in yellow.

1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051

Avalon Value Heuristics Function (After)

```
def evaluate_state(state: dict) -> tuple[dict[int, float], dict]:  
    prob_good_victory = 0.5 # probability that Good will win  
  
    if phase == 0: # team selection phase  
        num_successful quests = sum(historical_quest_results)  
        num_failed_quests = len(historical_quest_results) - num_successful_quests  
        num_remaining_quests = len(num_participants_per_quest) - len(historical_quest_results)  
        num_evil = len(players) - num_good  
        num_evil_in_quest_team = len([player for player in quest_team if not is_good[player]])  
  
        if num_successful_quests >= 3:  
            prob_good_victory = 0.9  
        elif num_failed_quests >= 3:  
            prob_good_victory = 0.1  
  
        intermediate_value1 = {'num_successful_quests': num_successful_quests, 'num_failed_quests':  
            num_failed_quests,  
            'num_remaining_quests': num_remaining_quests}  
        intermediate_value2 = {'num_evil_in_quest_team': num_evil_in_quest_team}  
  
    elif phase == 1: # team approval phase  
        num_evil_in_quest_team = len([player for player in quest_team if not is_good[player]])  
  
        prob_good_victory = 0.8 if num_evil_in_quest_team == 0 else 0.2  
  
        intermediate_value1 = {'num_evil_in_quest_team': num_evil_in_quest_team}  
        intermediate_value2 = {} # Initialize empty dictionary to avoid the error  
  
    elif phase == 2: # quest phase  
        num_successful_quests = sum(historical_quest_results)  
        num_failed_quests = len(historical_quest_results) - num_successful_quests  
  
        prob_good_victory = 0.9 if num_successful_quests > num_failed_quests else 0.1  
  
        intermediate_value1 = {'num_successful_quests': num_successful_quests, 'num_failed_quests':  
            num_failed_quests}  
        intermediate_value2 = {} # Initialize empty dictionary to avoid the error  
  
    elif phase == 3: # assassination phase  
        merlin_index = roles.index('Merlin')  
        assassin_index = roles.index('Assassin')  
  
        if assassin_index in quest_team:  
            prob_good_victory = 0.1  
        else:  
            prob_good_victory = 0.9  
  
        intermediate_value1 = {'merlin_index': merlin_index, 'assassin_index': assassin_index}  
        intermediate_value2 = {} # Initialize empty dictionary to avoid the error  
  
        expected_winrates_per_player = dict()  
        prob_evil_victory = 1 - prob_good_victory  
        for player in players:  
            if is_good[player]:  
                expected_winrates_per_player[player] = prob_good_victory  
            else:  
                expected_winrates_per_player[player] = prob_evil_victory  
  
        intermediate_values = {'intermediate_value1': intermediate_value1, 'intermediate_value2':  
            intermediate_value2}  
  
    return expected_winrates_per_player, intermediate_values
```

2052 H.3.3 EXAMPLE 3

2053

2054

2055

2056

2057

2058 Below is the GOPS Value Heuristics Function before the strategy improvement.

2059

2060

2061

2062

2063

2064

2065

2066

2067

GOPS Value Heuristics Function (Before)

2068

2069

2070

2071

2072

2073

2074

2075

2076

2077

2078

2079

2080

2081

2082

2083

2084

2085

2086

2087

2088

2089

2090

2091

2092

2093

2094

2095

2096

2097

2098

2099

2100

2101

2102

2103

2104

2105

```
def evaluate_state(state) -> tuple[tuple[float, float], dict]:
    score_cards = state[0]
    player_0_played_cards = state[1]
    player_1_played_cards = state[2]
    is_turn = state[3]
    player_0_score = state[4]
    player_1_score = state[5]
    score_deck = state[6]
    player_0_hand = state[7]
    player_1_hand = state[8]

    # Calculate initial potentials
    player_0_potential = sum(player_0_hand)
    player_1_potential = sum(player_1_hand)
    score_potential = sum(score_deck)

    # Update player potentials based on remaining cards and score deck
    player_0_potential += sum(card for card in player_0_hand if any(card > score for score in score_deck
    ))
    player_1_potential += sum(card for card in player_1_hand if any(card > score for score in score_deck
    ))

    # Add half of the score potential to the player who has the turn
    if is_turn:
        player_0_potential += score_potential / 2
    else:
        player_1_potential += score_potential / 2

    # Count the number of certain wins for each player
    player_0_certain_wins = sum(card > max(player_1_hand) for card in player_0_hand)
    player_1_certain_wins = sum(card > max(player_0_hand) for card in player_1_hand)

    rounds_left = len(score_deck)

    # Dynamic adjustment based on specific cards played
    player_0_certain_wins_adjust = 0
    player_1_certain_wins_adjust = 0
    for i in range(len(player_0_played_cards)):
        if player_0_played_cards[i] > player_1_played_cards[i]:
            player_0_certain_wins_adjust += 1
        elif player_1_played_cards[i] > player_0_played_cards[i]:
            player_1_certain_wins_adjust += 1

    player_0_certain_wins += player_0_certain_wins_adjust
    player_1_certain_wins += player_1_certain_wins_adjust

    # Add potential scores from certain wins
    if rounds_left <= player_0_certain_wins:
        highest_scores = sorted(score_deck)[-rounds_left:]
        player_0_potential += sum(highest_scores)

    if rounds_left <= player_1_certain_wins:
        highest_scores = sorted(score_deck)[-rounds_left:]
        player_1_potential += sum(highest_scores)
```

```

2106
2107
2108 # Calculate expected scores
2109 player_0_expected_score = player_0_score + player_0_potential
2110 player_1_expected_score = player_1_score + player_1_potential
2111
2112 # Calculate the difference in the sum of played cards
2113 sum_player_0_played_cards = sum(player_0_played_cards)
2114 sum_player_1_played_cards = sum(player_1_played_cards)
2115 sum_played_cards_diff = sum_player_0_played_cards - sum_player_1_played_cards
2116
2117 # Adjust scores based on the difference in the sum of played cards
2118 adjustment_factor = 0.1 # Adjust this factor based on the level of adjustment desired
2119 player_0_expected_score += sum_played_cards_diff * adjustment_factor
2120 player_1_expected_score -= sum_played_cards_diff * adjustment_factor
2121
2122 # Dynamic adjustment based on uncertainty in the game state
2123 uncertainty_factor = 0.1 * rounds_left # Adjust this factor based on the level of uncertainty
2124 desired
2125 player_0_expected_score += player_0_potential * uncertainty_factor
2126 player_1_expected_score += player_1_potential * uncertainty_factor
2127
2128 # Refine uncertainty factor based on specific conditions
2129 if len(player_0_hand) == 1:
2130     uncertainty_factor *= 0.5
2131 if len(player_1_hand) == 1:
2132     uncertainty_factor *= 0.5
2133
2134 # Store intermediate values in a dictionary
2135 intermediate_values = {
2136     'player_0_potential': player_0_potential,
2137     'player_1_potential': player_1_potential,
2138     'score_potential': score_potential,
2139     'player_0_certain_wins': player_0_certain_wins,
2140     'player_1_certain_wins': player_1_certain_wins,
2141     'player_0_certain_wins_adjust': player_0_certain_wins_adjust,
2142     'player_1_certain_wins_adjust': player_1_certain_wins_adjust,
2143     'sum_played_cards_diff': sum_played_cards_diff,
2144     'uncertainty_factor': uncertainty_factor
2145 }
2146
2147 return (player_0_expected_score, player_1_expected_score), intermediate_values
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159

```

Below is the selected idea to improve the above strategy.

Selected GOPS Value Heuristics Improvement Idea

Incorporate a term in the value function that takes into account the distribution of score cards left in the deck. Certain score cards may have a higher value or strategic importance, so adjusting the potentials based on the presence of these cards can provide a more accurate evaluation of the state.

Below is the GOPS Value Heuristics Function after enhancement by LLM. The improved parts by LLM are highlighted in yellow.

2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213

GOPS Value Heuristics Function (After)

```
def evaluate_state(state) -> tuple[tuple[float, float], dict]:
    score_cards = state[0]
    player_0_played_cards = state[1]
    player_1_played_cards = state[2]
    is_turn = state[3]
    player_0_score = state[4]
    player_1_score = state[5]
    score_deck = state[6]
    player_0_hand = state[7]
    player_1_hand = state[8]

    # Calculate initial potentials
    player_0_potential = sum(player_0_hand)
    player_1_potential = sum(player_1_hand)
    score_potential = sum(score_deck)

    # Update player potentials based on remaining cards and score deck
    player_0_potential += sum(card for card in player_0_hand if any(card > score for score in score_deck
    ))
    player_1_potential += sum(card for card in player_1_hand if any(card > score for score in score_deck
    ))

    # Add half of the score potential to the player who has the turn
    if is_turn:
        player_0_potential += score_potential / 2
    else:
        player_1_potential += score_potential / 2

    # Count the number of certain wins for each player
    player_0_certain_wins = sum(card > max(player_1_hand) for card in player_0_hand)
    player_1_certain_wins = sum(card > max(player_0_hand) for card in player_1_hand)

    rounds_left = len(score_deck)

    # Dynamic adjustment based on specific cards played
    player_0_certain_wins_adjust = 0
    player_1_certain_wins_adjust = 0
    for i in range(len(player_0_played_cards)):
        if player_0_played_cards[i] > player_1_played_cards[i]:
            player_0_certain_wins_adjust += 1
        elif player_1_played_cards[i] > player_0_played_cards[i]:
            player_1_certain_wins_adjust += 1

    player_0_certain_wins += player_0_certain_wins_adjust
    player_1_certain_wins += player_1_certain_wins_adjust

    # Add potential scores from certain wins
    if rounds_left <= player_0_certain_wins:
        highest_scores = sorted(score_deck)[-rounds_left:]
        player_0_potential += sum(highest_scores)

    if rounds_left <= player_1_certain_wins:
        highest_scores = sorted(score_deck)[-rounds_left:]
        player_1_potential += sum(highest_scores)

    # Incorporate distribution of score cards left in the deck
    important_score_cards = {7, 10, 13} # Example: define important score cards
    player_0_potential += sum(card for card in player_0_hand if card in important_score_cards)
    player_1_potential += sum(card for card in player_1_hand if card in important_score_cards)

    # Calculate expected scores
    player_0_expected_score = player_0_score + player_0_potential
    player_1_expected_score = player_1_score + player_1_potential

    # Calculate the difference in the sum of played cards
    sum_player_0_played_cards = sum(player_0_played_cards)
    sum_player_1_played_cards = sum(player_1_played_cards)
    sum_played_cards_diff = sum_player_0_played_cards - sum_player_1_played_cards
```

```

2214
2215
2216     # Adjust scores based on the difference in the sum of played cards
2217     adjustment_factor = 0.1 # Adjust this factor based on the level of adjustment desired
2218     player_0_expected_score += sum_played_cards_diff * adjustment_factor
2219     player_1_expected_score -= sum_played_cards_diff * adjustment_factor
2220
2221     # Dynamic adjustment based on uncertainty in the game state
2222     uncertainty_factor = 0.1 * rounds_left # Adjust this factor based on the level of uncertainty
2223     desired
2224     player_0_expected_score += player_0_potential * uncertainty_factor
2225     player_1_expected_score += player_1_potential * uncertainty_factor
2226
2227     # Refine uncertainty factor based on specific conditions
2228     if len(player_0_hand) == 1:
2229         uncertainty_factor *= 0.5
2230     if len(player_1_hand) == 1:
2231         uncertainty_factor *= 0.5
2232
2233     # Store intermediate values in a dictionary
2234     intermediate_values = {
2235         'player_0_potential': player_0_potential,
2236         'player_1_potential': player_1_potential,
2237         'score_potential': score_potential,
2238         'player_0_certain_wins': player_0_certain_wins,
2239         'player_1_certain_wins': player_1_certain_wins,
2240         'player_0_certain_wins_adjust': player_0_certain_wins_adjust,
2241         'player_1_certain_wins_adjust': player_1_certain_wins_adjust,
2242         'sum_played_cards_diff': sum_played_cards_diff,
2243         'uncertainty_factor': uncertainty_factor
2244     }
2245
2246     return (player_0_expected_score, player_1_expected_score), intermediate_values
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258 H.3.4 EXAMPLE 4
2259
2260
2261
2262
2263
2264
2265
2266
2267

```

Below is the GOPS Value Heuristics Function before the strategy improvement.

2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321

GOPS Value Heuristics Function (Before)

```
def evaluate_state(state) -> tuple[tuple[float, float], dict]:
    score_cards = state[0]
    player_0_played_cards = state[1]
    player_1_played_cards = state[2]
    is_turn = state[3]
    player_0_score = state[4]
    player_1_score = state[5]
    score_deck = state[6]
    player_0_hand = state[7]
    player_1_hand = state[8]

    # Calculate initial potentials
    player_0_potential = sum(player_0_hand)
    player_1_potential = sum(player_1_hand)
    score_potential = sum(score_deck)

    # Add half of the score potential to the player who has the turn
    if is_turn:
        player_0_potential += score_potential / 2
    else:
        player_1_potential += score_potential / 2

    # Count the number of certain wins for each player
    player_0_certain_wins = sum(card > max(player_1_hand) for card in player_0_hand)
    player_1_certain_wins = sum(card > max(player_0_hand) for card in player_1_hand)

    rounds_left = len(score_deck)

    # Add potential scores from certain wins
    if rounds_left <= player_0_certain_wins:
        highest_scores = sorted(score_deck)[-rounds_left:]
        player_0_potential += sum(highest_scores)

    if rounds_left <= player_1_certain_wins:
        highest_scores = sorted(score_deck)[-rounds_left:]
        player_1_potential += sum(highest_scores)

    # New improvement: Incorporate a probabilistic approach based on the remaining score cards
    player_0_expected_score = player_0_score + player_0_potential
    player_1_expected_score = player_1_score + player_1_potential

    # Dynamic evaluation of opponent's potential moves
    opponent_potential = sum(player_0_hand) if is_turn else sum(player_1_hand)
    opponent_certain_wins = sum(card > max(player_1_hand) for card in player_0_hand) if is_turn else sum(
        card > max(player_0_hand) for card in player_1_hand)

    # Adjust evaluation based on opponent's potential plays
    if rounds_left <= opponent_certain_wins:
        highest_scores = sorted(score_deck)[-rounds_left:]
        if is_turn:
            player_1_potential += sum(highest_scores)
        else:
            player_0_potential += sum(highest_scores)

    # Store intermediate values in a dictionary
    intermediate_values = {
        'player_0_potential': player_0_potential,
        'player_1_potential': player_1_potential,
        'score_potential': score_potential,
        'player_0_certain_wins': player_0_certain_wins,
        'player_1_certain_wins': player_1_certain_wins,
        'opponent_potential': opponent_potential,
        'opponent_certain_wins': opponent_certain_wins,
        'player_0_expected_score': player_0_expected_score,
        'player_1_expected_score': player_1_expected_score
    }

    return (player_0_expected_score, player_1_expected_score), intermediate_values
```

Below is the selected idea to improve the above strategy.

2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375

Selected GOPS Value Heuristics Improvement Idea

Introduce a penalty factor for potential scores that are excessively high, indicating potential overestimation. By capping the potential scores or applying a scaling factor based on certain thresholds, we can prevent the function from assigning unrealistic values and improve the accuracy of the value heuristic.

Below is the GOPS Value Heuristics Function after enhancement by LLM. The improved parts by LLM are highlighted in yellow.

```
GOPS Value Heuristics Function (After)

def evaluate_state(state) -> tuple[tuple[float, float], dict]:
    score_cards = state[0]
    player_0_played_cards = state[1]
    player_1_played_cards = state[2]
    is_turn = state[3]
    player_0_score = state[4]
    player_1_score = state[5]
    score_deck = state[6]
    player_0_hand = state[7]
    player_1_hand = state[8]

    # Calculate initial potentials
    player_0_potential = sum(player_0_hand)
    player_1_potential = sum(player_1_hand)
    score_potential = sum(score_deck)

    # Add half of the score potential to the player who has the turn
    if is_turn:
        player_0_potential += score_potential / 2
    else:
        player_1_potential += score_potential / 2

    # Count the number of certain wins for each player
    player_0_certain_wins = sum(card > max(player_1_hand) for card in player_0_hand)
    player_1_certain_wins = sum(card > max(player_0_hand) for card in player_1_hand)

    rounds_left = len(score_deck)

    # Add potential scores from certain wins
    if rounds_left <= player_0_certain_wins:
        highest_scores = sorted(score_deck)[-rounds_left:]
        player_0_potential += sum(highest_scores)

    if rounds_left <= player_1_certain_wins:
        highest_scores = sorted(score_deck)[-rounds_left:]
        player_1_potential += sum(highest_scores)
```

```

2376
2377
2378 # New improvement: Incorporate a probabilistic approach based on the remaining score cards
2379 player_0_expected_score = player_0_score + player_0_potential
2380 player_1_expected_score = player_1_score + player_1_potential
2381
2382 # Dynamic evaluation of opponent's potential moves
2383 opponent_potential = sum(player_0_hand) if is_turn else sum(player_1_hand)
2384 opponent_certain_wins = sum(card > max(player_1_hand) for card in player_0_hand) if is_turn else sum
2385 (card > max(player_0_hand) for card in player_1_hand)
2386
2387 # Adjust evaluation based on opponent's potential plays
2388 if rounds_left <= opponent_certain_wins:
2389     highest_scores = sorted(score_deck)[-rounds_left:]
2390     if is_turn:
2391         player_1_potential += sum(highest_scores)
2392     else:
2393         player_0_potential += sum(highest_scores)
2394
2395 # Introduce a penalty factor for excessively high potential scores
2396 penalty_threshold = 100 # Define a threshold for potential scores to trigger penalty
2397 penalty_factor = 0.5 # Define a factor by which to reduce potential scores above threshold
2398
2399 if player_0_potential > penalty_threshold:
2400     player_0_potential = penalty_threshold + (player_0_potential - penalty_threshold) *
2401     penalty_factor
2402
2403 if player_1_potential > penalty_threshold:
2404     player_1_potential = penalty_threshold + (player_1_potential - penalty_threshold) *
2405     penalty_factor
2406
2407 # Store intermediate values in a dictionary
2408 intermediate_values = {
2409     'player_0_potential': player_0_potential,
2410     'player_1_potential': player_1_potential,
2411     'score_potential': score_potential,
2412     'player_0_certain_wins': player_0_certain_wins,
2413     'player_1_certain_wins': player_1_certain_wins,
2414     'opponent_potential': opponent_potential,
2415     'opponent_certain_wins': opponent_certain_wins,
2416     'player_0_expected_score': player_0_expected_score,
2417     'player_1_expected_score': player_1_expected_score
2418 }
2419
2420 return (player_0_expected_score, player_1_expected_score), intermediate_values
2421
2422
2423
2424
2425
2426
2427
2428
2429

```

2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483

I DIALOGUE GUIDE LLM PROMPT AND OUTPUT EXAMPLES

This sections shows the system prompts of dialogue guidance on LLM and several examples, including system prompts, idea generation prompts, and strategy implementation examples.

I.1 SYSTEM PROMPTS

Below is the Dialogue guide system prompt.

Dialogue guide system prompt

You are a coach trying to write a section of a strategy guide on how to play a game well.

The specific section of the strategy guide you are writing right now is on how to play the Merlin role effectively during the discussion phase so that they can win the game. Recall that players often use the discussion phase to (1) gather information about other players, (2) try to convince other players of their innocence or guilt, and (3) try to persuade other players of a particular course of action. The game you are interested in is called The Resistance: Avalon. The Resistance: Avalon is the game of hidden identities and social deduction. There are two teams in the game: Good and Evil. Each player has a hidden identity (role) and side.

Dialogue guide signature follows, specifying the structures and contents dialogues regarding role playing the Merlin.

Dialogue guide signature

Your guide should be in the form of a worksheet that the student can use to build their speech. You should order the worksheet questions in a way that makes logical sense, and you should have no more than six questions. Your questions should instruct the reader to write parts of their speech.

The title of your section should be “Questions to fill out before speaking as the Merlin role”. Below is an example of how your worksheet should look like:

1. Questions to fill out before speaking as the Merlin role
2. Q1: Which player seems the most suspicious of you and why?
3. Q2: For the player that seems the most suspicious of you, produce a statement addressing their suspicious.
4. Q3: Which player is the quest leader?
5. Q4: Produce a statement addressing the quest leader to convince them to support your intended course of action/ desired team.
6. Q5: Which player is the most supportive of you?
7. Q6: Produce a statement addressing the supportive player to convince them to support your intended course of action/ desired team.

2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537

Example Generated Dialogue Strategy Guide for Assassin

1. Q1: Which player seems to have a deeper understanding of the game flow than normal Servants should possess?
2. Q2: Develop a non-confrontational statement to subtly challenge this player. This should cause them to either prove their innocence or reveal more clues about their identity.
3. Q3: Who has been the most influential in the team selection and voting process?
4. Q4: Devise a statement to express agreement with this player's viewpoint subtly. This should make you less suspicious while enabling you to manipulate the discussion.
5. Q5: Which player seems the most supportive of your views and actions in the game?
6. Q6: Craft a statement subtly emphasizing your alignment with this supportive player's thoughts. This should increase your chances of being included in quest teams and reduce suspicion around you.

I.2 IDEA GENERATION EXAMPLES

The following is the selected dialogue guide speaking as the Merlin role.

Select dialogue guide

Questions to fill out before speaking as the Merlin role

1. Who are the players you suspect are on the Evil side and why?
2. Draft a subtle statement that casts doubt on these players without directly accusing them.
3. Who is the current round leader and which players are you considering to suggest for the team?
4. Create a subtle suggestion that influences the leader's team decision towards your preferred team without revealing your knowledge.
5. What are the voting patterns you've observed and what do these patterns indicate about the players' identities?
6. Analyze the voting patterns and team compositions proposed by each player to identify potential alliances or suspicious behaviors. Craft a statement to subtly address these observations without revealing your role.
7. Which player seems the most supportive of Good and how can you align with them during the discussion phase to better ensure success for your team?

A feedback example is provided for better understanding of dialogue guidance to teach LLM how to role play Merlin.

2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591

Feedback example description

Example 2:

The student using your guide was playing as player 0 in this example filled out your guide as follows:

1. Q1: Based on the statements made so far, I suspect Players 2 and 4 are on the Evil side. Player 2's suggestion of a team consisting of Player 1 and Player 2 seems too convenient, possibly indicating a desire to control the team composition. Player 4's agreement with Player 1 and Player 2 without offering any alternative team suggestions raises suspicion as well.
2. Q2: "I can't help but wonder if some players are trying to steer the team selection process in a particular direction. It's important for us to carefully consider all perspectives and not be swayed by potentially self-serving suggestions."
3. Q3: The current round leader is Player 1. I would suggest a team consisting of Player 0 (Merlin) and Player 1, as this would allow me to closely observe the actions of the other players and potentially guide the team towards success.
4. Q4: "Player 1, I believe that a team comprised of Player 0 and Player 1 would provide a strong foundation for success. By working closely together, we can ensure that our mission is carried out effectively."
5. Q5: The voting patterns so far show agreement among Players 1, 2, and 4, indicating a potential alliance or shared goals. Player 3's suggestion of including themselves in the team may also indicate a desire to control the outcome of the quest.
6. Q6: "I've noticed a pattern of agreement among certain players in the team selection process. It's important for us to consider all perspectives and not be swayed by potentially biased suggestions. Let's focus on working together as a team to achieve our common goal."
7. Q7: Player 1 seems the most supportive of Good based on their consistent emphasis on unity and success for the Quests. I will align with Player 1 during the discussion phase to ensure that our team is selected and the quest is completed successfully.

Then they said the following during the discussion phase:

"Fellow players, as we embark on this quest, it is crucial that we approach the team selection process with caution and consideration. I have noticed a pattern of agreement among certain players, which could potentially indicate a desire to steer the outcome in a particular direction. It is important for us to carefully consider all perspectives and not be swayed by potentially biased suggestions.

I believe that a team comprised of Player 0 and Player 1 would provide a strong foundation for success. By working closely together, we can ensure that our mission is carried out effectively. Player 1 has consistently emphasized unity and success for the Quests, making them a trustworthy ally in this endeavor.

Let us focus on working together as a team to achieve our common goal, rather than allowing personal agendas to influence our decisions. Let us approach this quest with unity and determination, ensuring that Good prevails in the end. Thank you for your attention."

2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645

Below is some feedback from the other players on how player 0 performed during the discussion phase. Recall that Good players are trying to access which other players are likely to be Good, while Evil players are trying to access which Good player is likely to be Merlin.

Player 3 with role Servant had the following thoughts on this discussion round:

Thought:
Player 1's speech emphasizes unity and success for the Quests, which aligns with the goals of Good and decreases the probability of Player 1 being Evil. Player 2's statement also focuses on unity and careful consideration, indicating a commitment to the success of Good and decreasing the probability of Player 2 being Evil. Player 3's response as a Servant of Arthur is focused on the success of the Quests and unity, which aligns with the goals of Good and decreases the probability of Player 3 being Evil. Player 4's speech seems overly enthusiastic and focused on unity and cooperation, which could be a tactic to divert attention from their true identity as an Evil player. This raises suspicion and increases the probability of Player 4 being Evil.

Dictionary: {0: (-1, 'decreased slightly'), 1: (-1, 'decreased slightly'), 2: (0, 'stayed the same'), 3: (-2, 'decreased significantly'), 4: (2, 'increased significantly')}

Player 4 with role Assassin had the following thoughts on this discussion round:

Thought:
Player 2 seems to suspect the Evil players (Player 3 and Player 4) without any evidence and is trying to steer the conversation towards them. This aligns with the behavior of Merlin, who knows the identity of the Evil players and is trying to subtly guide the Good players towards the correct decisions. This increases the probability of Player 2 being Merlin. On the other hand, Player 3's speech seems generally clueless and lacks any strategic insight, which decreases the probability of Player 3 being Merlin.

Dictionary: {0: (0, 'stayed the same'), 1: (0, 'stayed the same'), 2: (2, 'increased significantly'), 3: (-2, 'decreased significantly'), 4: (-2, 'decreased significantly')}

This is the prompt for feedback reflections.

Prompt for feedback reflections

```
<System prompt>  
<Game rules>  
You previously generated the following section of the strategy guide: <Previous guide>  
Below is some feedback on how your guide performed when a student used it to play the  
game:  
<Feedback examples>  
Based on the feedback given and the guide section you generated previously, what are some  
conclusions you can draw from the feedback? Make sure to cite the specific examples in the  
feedback to justify your analysis.
```

The following are the feedback reflections generated by LLM.

2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699

Generated feedback reflections

In Example 0, Player 0 effectively followed the guide by suspecting certain players of being on the Evil side, crafting subtle statements to cast doubt on them, and aligning with players supportive of Good. This led to Player 0 being perceived positively by other players, with Player 2's probability of being Merlin decreasing significantly due to their behavior aligning more with the Evil side.

In Example 1, Player 0 again effectively followed the guide by suspecting players, making subtle statements, aligning with a supportive player, and emphasizing unity and success in their speech. This resulted in Player 0 being perceived positively by other players, with Player 4's probability of being Merlin decreasing significantly due to their lack of strategic insight.

In Example 2, Player 0 successfully identified suspicious behavior in other players, made subtle statements, and aligned with a supportive player. This led to Player 0 being perceived positively, with Player 2's probability of being Merlin decreasing significantly due to their behavior aligning more with the Evil side.

In Example 3, Player 0 effectively followed the guide by suspecting players, making subtle statements, aligning with a supportive player, and emphasizing unity. This resulted in Player 0 being perceived positively by other players, with Player 4's probability of being Merlin increasing significantly due to their behavior raising suspicion.

- Overall, the guide's emphasis on analyzing player behavior, making subtle statements, and aligning with supportive players has proven to be successful in helping players navigate the discussion phase effectively as the Merlin role in The Resistance: Avalon.
- Players who followed the guide were able to strategically assess the situation, cast doubt on potential Evil players, and align with Good players to increase their chances of success.

Below is the Prompt for idea generation regarding this dialogue guides

2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753

Prompt for idea generation

<System prompt>
<Game rules>
You previously generated the following section of the strategy guide:
<Previous guide>
Here is a possible way to improve your guide:
<Feedback reflections>

Based on the worksheet, feedback, and conclusions you drew, what are 2 improvements that you can make to the worksheet that you think will have the most impact?
Be as specific and concrete as possible, including what questions to add, edit, or remove, and write them out in the following format:

- Thoughts: <your thoughts here>
- Idea 1: <your idea here>
- Idea 2: <your idea here>
- ...

Here's an example of what this might look like for 3 improvement ideas:

- Thoughts: I should tell the reader to address each player individually in the guide.
- Idea 1: Add a question asking the reader who they think is most suspicious of them and produce a statement addressing their suspicions.
- Idea 2: Add a question asking the reader to consider which other player is most supportive of them and produce a statement addressing their support.
- Idea 3: Add a question asking the reader produce a statement addressed

Generated new improvement ideas are demonstrated below, showing the improvements made by LLM following the prompt for idea generation.

Generated new improvement ideas

Thoughts: The current worksheet focuses on analyzing player behavior, making subtle statements, and aligning with supportive players, which are all crucial strategies for playing the Merlin role effectively. To enhance the guide further, we can add questions that prompt the player to consider their own actions and statements more explicitly.

1. Idea 1: Add a question asking the player to reflect on how their proposed team composition would benefit the success of the Quest and provide a statement justifying their choices.
2. Idea 2: Add a question asking the player to assess their level of suspicion towards each player and craft a statement addressing how they will interact with those players during the discussion phase.
3. Idea 3: Add a question asking the player to consider how they can subtly guide the discussion towards a successful Quest.

I.3 STRATEGY IMPLEMENTATION EXAMPLES

Recall the strategy implementation mentioned in the paper before. Here, we implement strategy improvements on dialogue guide. The improved dialogue guide is highlighted.

I.3.1 EXAMPLE 1

Below is the dialogue guide before any improvements happened.

2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807

Dialogue guide (previous)

Questions to fill out before speaking as the Merlin role

1. Who are the players you suspect are on the Evil side and why?
2. Draft a subtle statement that casts doubt on these players without directly accusing them.
3. Who is the current round leader and which players are you considering to suggest for the team?
4. Create a subtle suggestion that influences the leader's team decision towards your preferred team without revealing your knowledge.
5. What are the voting patterns you've observed and what do these patterns indicate about the players' identities?
6. Analyze the voting patterns and team compositions proposed by each player to identify potential alliances or suspicious behaviors. Craft a statement to subtly address these observations without revealing your role.
7. Which player seems the most supportive of Good and how can you align with them during the discussion phase to better ensure success for your team?

Below is the improvement idea prompted into LLM.

Improvement idea

Add a question asking the player to assess their level of suspicion towards each player and craft a statement addressing how they will interact with those players during the discussion phase.

Here, implementation prompty follows to teach LLM how to improve the dialogue guide.

Implementation prompt

```
<System prompt>  
<Game rules>  
You previously generated the following section of the strategy guide:  
<Previous guide>  
Here is a possible way to improve your guide:  
<Improvement idea>  
Implement this improvement into the guide section as best as you can, but do not change the original guide too much. Make sure to stay within the scope of the guide section, which we reiterate below:  
<Guide signature>
```

Finally, we have an improved dialogue guide, where the changes are highlighted below comparing to the previous guide.

2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861

Dialogue guide (improved)

Questions to fill out before speaking as the Merlin role

1. Who are the players you suspect are on the Evil side and why?
2. Draft a subtle statement that casts doubt on these players without directly accusing them.
3. Who is the current round leader and which players are you considering to suggest for the team?
4. Create a subtle suggestion that influences the leader's team decision towards your preferred team without revealing your knowledge.
5. What are the voting patterns you've observed and what do these patterns indicate about the players' identities?
6. Analyze the voting patterns and team compositions proposed by each player to identify potential alliances or suspicious behaviors. Craft a statement to subtly address these observations without revealing your role.
7. Which player seems the most supportive of Good and how can you align with them during the discussion phase to better ensure success for your team?
8. Assess your level of suspicion towards each player and craft a statement addressing how you will interact with those players during the discussion.

I.3.2 EXAMPLE 2

Below is the dialogue guide before any improvements happened.

Dialogue guide (previous)

Questions to fill out before speaking as the Merlin role

1. Who are the players you suspect are on the Evil side and why?
2. Draft a subtle statement that casts doubt on these players without directly accusing them.
3. Who is the current round leader and which players are you considering to suggest for the team?
4. Create a subtle suggestion that influences the leader's team decision towards your preferred team without revealing your knowledge.
5. What are the voting patterns you've observed and what do these patterns indicate about the players' identities?
6. Craft a statement to share these observations subtly without giving away your role.

Below is the improvement idea prompted into LLM.

Improvement idea

Add a question asking the reader to identify which players they feel most aligned with in terms of strategy and decision-making, and how they can subtly align themselves with these players during the discussion phase.

Here, implementation prompt follows to teach LLM how to improve the dialogue guide.

2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2910
2911
2912
2913
2914
2915

Implementation prompt

<System prompt>
<Game rules>
You previously generated the following section of the strategy guide:
<Previous guide>
Here is a possible way to improve your guide:
<Improvement idea>
Implement this improvement into the guide section as best as you can, but do not change the original guide too much. Make sure to stay within the scope of the guide section, which we reiterate below:
<Guide signature>

Finally, we have a improved dialogue guide, where the changes are highlighted below comparing to the previous guide

Dialogue guide (improved)

Questions to fill out before speaking as the Merlin role

1. Who are the players you suspect are on the Evil side and why?
2. Draft a subtle statement that casts doubt on these players without directly accusing them.
3. Who is the current round leader and which players are you considering to suggest for the team?
4. Create a subtle suggestion that influences the leader’s team decision towards your preferred team without revealing your knowledge.
5. What are the voting patterns you’ve observed and what do these patterns indicate about the players’ identities?
6. Craft a statement to share these observations subtly without giving away your role.
7. Which player seems the most aligned with your strategic decisions and why?
8. Produce a statement to subtly align yourself with the player you feel most aligned with during the discussion phase.

I.3.3 EXAMPLE 3

Below is the dialogue guide before any improvements happened.

2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969

Dialogue guide (previous)

Questions to fill out before speaking as the Merlin role

1. Who are the players you suspect are on the Evil side and why?
2. Draft a subtle statement that casts doubt on these players without directly accusing them.
3. Who is the current round leader and which players are you considering to suggest for the team?
4. Create a subtle suggestion that influences the leader's team decision towards your preferred team without revealing your knowledge.
5. What are the voting patterns you've observed and what do these patterns indicate about the players' identities?
6. Craft a statement to share these observations subtly without giving away your role.
7. Which player seems the most aligned with your strategic decisions and why?
8. Produce a statement to subtly align yourself with the player you feel most aligned with during the discussion phase.

Below is the improvement idea prompted into LLM.

Improvement idea

Add a question asking the reader to analyze the voting patterns and craft a statement subtly highlighting any inconsistencies or suspicious trends in the voting behavior of specific players.

Here, implementation prompt follows to teach LLM how to improve the dialogue guide.

Implementation prompt

```
<System prompt>  
<Game rules>  
You previously generated the following section of the strategy guide:  
<Previous guide>  
Here is a possible way to improve your guide:  
<Improvement idea>  
Implement this improvement into the guide section as best as you can, but do not change the original guide too much. Make sure to stay within the scope of the guide section, which we reiterate below:  
<Guide signature>
```

Finally, we have an improved dialogue guide, where the changes are highlighted below comparing to the previous guide

2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2999
3000
3001
3002
3003
3004
3005
3006
3007
3008
3009
3010
3011
3012
3013
3014
3015
3016
3017
3018
3019
3020
3021
3022
3023

Dialogue guide (improved)

Questions to fill out before speaking as the Merlin role

1. Who are the players you suspect are on the Evil side and why?
2. Draft a subtle statement that casts doubt on these players without directly accusing them.
3. Who is the current round leader and which players are you considering to suggest for the team?
4. Create a subtle suggestion that influences the leader's team decision towards your preferred team without revealing your knowledge.
5. What are the voting patterns you've observed and what do these patterns indicate about the players' identities?
6. Craft a statement to share these observations subtly without giving away your role.
7. Which player seems the most aligned with your strategic decisions and why?
8. Produce a statement to subtly align yourself with the player you feel most aligned with during the discussion phase.
9. Analyze the voting patterns and craft a statement subtly highlighting any inconsistencies or suspicious trends in the voting behavior of specific players.

3024
3025
3026
3027
3028
3029
3030
3031
3032
3033
3034
3035
3036
3037
3038
3039
3040
3041
3042
3043
3044
3045
3046
3047
3048
3049
3050
3051
3052
3053
3054
3055
3056
3057
3058
3059
3060
3061
3062
3063
3064
3065
3066
3067
3068
3069
3070
3071
3072
3073
3074
3075
3076
3077

J DIALOGUE GENERATION AND ANALYSIS EXAMPLE LLM INPUTS AND OUTPUTS

This sections shows the examples of dialogue generation and analysis, including descriptive prompts, dialogue generation, and dialogue analysis.

J.1 DESCRIPTIVE PROMPTS

Below is the discussion history prompted into LLM for better knowledge of dialogue generation.

Discussion history

Here is a summary of previous rounds of discussion so far:

- Player 1: "Greetings, fellow players. As a Servant of Arthur, my primary goal is to ensure the success of our Quests. I am committed to working together with all of you to achieve victory for Good. I believe that for this first Quest, a team consisting of Player 1 and Player 4 would be a strong choice. I hope for your support in approving this team, as I am confident that we can work effectively together to accomplish our goal. I encourage everyone to share their thoughts and strategies openly. Let's work together to assess each other's intentions and make the best decisions for the success of our Quest. Remember, the fate of Camelot rests in our hands, and we must be vigilant against the forces of Evil."
- Player 2: "Greetings, noble companions. I echo Player 1's sentiment in working together for the success of our Quests. I believe it is essential to carefully consider our team choices to ensure victory for Good. I suggest that for this first Quest, a team consisting of Player 1 and Player 2 would provide a strong foundation for success. Together, we can overcome any challenges that come our way. Let us unite in our efforts and show our unwavering dedication to the cause. I urge all players to carefully evaluate each team proposal and remember that unity is key to thwarting the forces of Evil."
- Player 3: "Greetings, noble companions. I echo the sentiments shared by Player 1 and Player 2 regarding our shared goal of ensuring the success of our Quests. Unity and cooperation will be key in our endeavors to protect Camelot from the forces of Evil. As we deliberate on the composition of our team for this first Quest, I would like to offer a perspective that considers the balance and strength of our choices. A team comprised of Player 1 and Player 3 would bring together a diverse set of skills and viewpoints that can greatly contribute to our success. By including Player 3 in the team, we can ensure a comprehensive approach to overcoming any challenges that may arise. Let us work together in harmony, leveraging our individual strengths for the greater good of Camelot. I urge my fellow players to consider this team proposal thoughtfully, as I believe it holds great promise for the success of our Quest."
- Player 4: "Greetings, honorable players. I appreciate the dedication and commitment displayed by Player 1, Player 2, and Player 3 towards our common goal of ensuring the success of this Quest. Unity and cooperation are indeed vital in our quest to safeguard Camelot. I believe that the proposed team of Player 1 and Player 2 presents a strong foundation for success. Their combined skills and perspectives can greatly benefit our mission, and I urge all players to consider this team thoughtfully for the betterment of our cause. Let us unite our strengths and work together seamlessly to overcome any challenges that may arise. By standing united, we can ensure the triumph of Good over the forces of Evil. I urge all players to support the team of Player 1 and Player 2 for the success of this Quest."

State description shows LLM which state it will be for role playing and dialogue generation.

3078
3079
3080
3081
3082
3083
3084
3085
3086
3087
3088
3089
3090
3091
3092
3093
3094
3095
3096
3097
3098
3099
3100
3101
3102
3103
3104
3105
3106
3107
3108
3109
3110
3111
3112
3113
3114
3115
3116
3117
3118
3119
3120
3121
3122
3123
3124
3125
3126
3127
3128
3129
3130
3131

State description

You are Player 0, with identity Merlin. You are on the side of Good. The Evil players are Players 2 and 4. Please do not forget your identity throughout the game.

The current state of the game is as follows:

- The number of players in the game is: 5
- This is the quest number 0 which requires 2 players and 1 fails to fail
- This is the 0 round of discussion
- The previous results for the quest were () (True for Success, False for Fail)
- The current phase of the game is the team selection phase
- The current leader is player 1

You would like the following team to be approved: [0, 1]

J.2 DIALOGUE GENERATION

Below is the prompt for filling out worksheet. Filling out this worksheet assists LLM generated better dialogue aligned with the role, Merlin as in the example.

Prompt for filling out worksheet

<System prompt>
<Game rules>
<Discussion history>
<State description>

Given the discussion history, current state, and your intended action, answer the above worksheet to the best of your ability. The answers should be based on the information you have and your own reasoning.

Six questions to fill out before speaking as the Merlin role

Q1: Who are the players you suspect are on the Evil side and why?

Q2: Draft a subtle statement that casts doubt on these players without directly accusing them.

Q3: Who is the current round leader and which players are you considering to suggest for the team?

Q4: Create a subtle suggestion that influences the leader’s team decision towards your preferred team without revealing your knowledge.

Q5: What are the voting patterns you’ve observed and what do these patterns indicate about the players’ identities?

Q6: Craft a statement to share these observations subtly without giving away your role.

The worksheet filled out by LLM follows, demonstrating the LLM capability and the effectiveness of prompting.

3132
3133
3134
3135
3136
3137
3138
3139
3140
3141
3142
3143
3144
3145
3146
3147
3148
3149
3150
3151
3152
3153
3154
3155
3156
3157
3158
3159
3160
3161
3162
3163
3164
3165
3166
3167
3168
3169
3170
3171
3172
3173
3174
3175
3176
3177
3178
3179
3180
3181
3182
3183
3184
3185

LLM filled out worksheet

Q1: I suspect Players 2 and 4 are on the Evil side because they both proposed the same team composition of Player 1 and Player 2, which seems too coordinated and potentially trying to exclude other players from the team.

Q2: "I've noticed a strong push for the same team composition from certain players, which may raise questions about their motivations and potential alliances within the group."

Q3: The current round leader is Player 1. I would suggest Player 0 and Player 1 for the team, as I believe this composition would provide a strong foundation for success and help me subtly guide the decision towards my preferred team.

Q4: "Player 1, I have a feeling that selecting Player 0 and Player 1 for the team could bring a diverse set of skills and viewpoints that may greatly contribute to our success. Let's consider all options carefully for the betterment of our cause."

Q5: The voting patterns have shown a strong push for the team composition proposed by Players 1 and 2, indicating potential coordination or alignment between these players. Player 3's suggestion of including themselves in the team also raises suspicions of wanting to secure a spot on the quest.

Q6: "I've noticed a pattern in the team composition suggestions and voting behaviors that may reveal certain alliances or motivations within the group. It's important for us to carefully evaluate all options and consider the implications for the success of our quest."

Below is the prompt for speech generation. Worksheet completion assists LLM to understand the states and role. The following prompt asks LLM to assemble responses to the questionnaire filled out above.

Prompt for speech generation

<System prompt>
<Filled out worksheet>

Assemble your responses to the questionnaire you just answered into a speech that would help you achieve your intent. Note that dialogue will be seen by all players in the game, so you should not reveal your identity.

Your speech:

Finally we have LLM generated speech after the preparations.

LLM generated speech

I've noticed a strong push for the same team composition from certain players, which may raise questions about their motivations and potential alliances within the group. The voting patterns have shown a strong push for the team composition proposed by Players 1 and 2, indicating potential coordination or alignment between these players. Player 3's suggestion of including themselves in the team also raises suspicions of wanting to secure a spot on the quest. It's important for us to carefully evaluate all options and consider the implications for the success of our quest. Let's consider all options carefully for the betterment of our cause. I have a feeling that selecting Player 0 and Player 1 for the team could bring a diverse set of skills and viewpoints that may greatly contribute to our success. As the current round leader, Player 1, I would suggest Player 0 and Player 1 for the team, as I believe this composition would provide a strong foundation for success and help me subtly guide the decision towards my preferred team. Thank you for considering my input.

3186 J.3 DIALOGUE ANALYSIS
3187

3188 After dialogue generation, we continue have LLM assume the role of Merlin, and dialogue analysis is
3189 conducted.
3190

3191 Below is the prompt for dialogue analysis regarding to the role of Merlin. A thought is included to
3192 guide LLM analyze.
3193

3194 Prompt for dialogue analysis: Merlin
3195

3196 <System prompt>
3197 <Game rules>
3198 <Discussion history>
3199 <State description>
3200

3201 Based on the discussions this round, for each player in players 0, 1, 2, 3, 4, do you think the
3202 probability of the player being Merlin (2) increased significantly (1) increased slightly (0)
3203 stayed the same (-1) decreased slightly or (-2) decreased significantly and why? Begin with
3204 your reasoning and end with a dictionary of player to your multiple choice answer, i.e. {0: (2,
3205 'increased significantly'), ..., 4: (0, 'stayed the same')}. Make sure you include quotes around
3206 the dictionary values that are strings.

3207 Here's an example:
3208

3209 Thought:

3210 Player 2 seems to suspect the Evil players (Player 3 and Player 4) without any evidence and
3211 is trying to steer the conversation towards them. This aligns with the behavior of Merlin,
3212 who knows the identity of the Evil players and is trying to subtly guide the Good players
3213 towards the correct decisions. This increases the probability of Player 2 being Merlin. On the
3214 other hand, Player 4's speech seems generally clueless and lacks any strategic insight, which
3215 decreases the probability of Player 4 being Merlin.

3216 Dictionary:

3217 {0: (0, 'stayed the same'), 1: (0, 'stayed the same'), 2: (2, 'increased significantly'), 3: (-2,
3218 'decreased significantly'), 4: (0, 'stayed the same')}

3220
3221 Generated dialogue analysis follows. The role is Merlin.
3222

3223
3224 LLM generated dialogue analysis: Merlin
3225

3226 Thought:

3227 Player 2 is actively participating in the discussion and proposing team compositions, which is
3228 a behavior that could align with the role of Merlin, who knows the Evil players and wants to
3229 guide the Good players towards success. This increases the probability of Player 2 being
3230 Merlin. Player 4, on the other hand, seems to be going along with the proposed teams without
3231 much strategic insight, which decreases the probability of Player 4 being Merlin.
3232

3233 Dictionary:

3234 {0: (0, 'stayed the same'), 1: (0, 'stayed the same'), 2: (2, 'increased significantly'), 3: (0,
3235 'stayed the same'), 4: (-2, 'decreased significantly')}

3236
3237
3238 Under dialouge analysis, besides focusing on one specific role, we also have LLM target on good and
3239 evil in the game, which is broader in terms of role playing. Below is the prompt for dialogue analysis
regarding to Good and Evil.

3240
3241
3242
3243
3244
3245
3246
3247
3248
3249
3250
3251
3252
3253
3254
3255
3256
3257
3258
3259
3260
3261
3262
3263
3264
3265
3266
3267
3268
3269
3270
3271
3272
3273
3274
3275
3276
3277
3278
3279
3280
3281
3282
3283
3284
3285
3286
3287
3288
3289
3290
3291
3292
3293

Prompt for dialogue analysis: Good and Evil

<System prompt>
<Game rules>
<Discussion history>
<State description>

Based on the discussions this round, for each player in players 0, 1, 2, 3, 4, do you think the probability of the player being Evil (2) increased significantly (1) increased slightly (0) stayed the same (-1) decreased slightly or (-2) decreased significantly and why? Begin with your reasoning and end with a dictionary of player to your multiple choice answer, i.e. {0: (2, 'increased significantly'), ..., 4: (0, 'stayed the same')}. Make sure you include quotes around the dictionary values that are strings.

Here's an example:

Thought: Player 4's speech seems overly enthusiastic and focused on unity and cooperation, which could be a tactic to divert attention from their true identity as an Evil player. This raises suspicion and increases the probability of Player 4 being Evil. On the other hand, Player 3's response as a Servant of Arthur is more focused on the success of the Quests and unity, which aligns with the goals of Good and decreases the probability of Player 3 being Evil.

Dictionary: {0: (0, 'stayed the same'), 1: (0, 'stayed the same'), 2: (2, 'increased significantly'), 3: (-2, 'decreased significantly'), 4: (0, 'stayed the same')}

Below is the dialogue analysis generated by LLM on Good and Evil.

LLM generated dialogue analysis: Good and Evil

Thought:
Player 1's speech is aligned with the goals of Good and focuses on ensuring the success of the Quests, which decreases the probability of Player 1 being Evil. Player 2's speech also emphasizes unity and cooperation, which is in line with the goals of Good and decreases the probability of Player 2 being Evil. Player 3's response suggests including themselves in the team, which may raise suspicions of wanting to secure a spot on the quest, potentially increasing the probability of Player 3 being Evil. Player 4's speech seems overly enthusiastic and focused on unity and cooperation, which could be a tactic to divert attention from their true identity as an Evil player, increasing the probability of Player 4 being Evil. Player 0's speech suggests a different team composition that may bring a diverse set of skills and viewpoints, potentially decreasing the probability of Player 0 being Evil.

Dictionary:
{0: (-1, 'decreased slightly'), 1: (-1, 'decreased slightly'), 2: (1, 'increased slightly'), 3: (1, 'increased slightly'), 4: (2, 'increased significantly')}

3294 K OTHER ABLATION STUDIES

3295

3296 K.0.1 BASE LLM MODEL

3297

3298 We show the performance of different base models in Figure 16.

3299

3300

3301

3302

3303

3304

3305

3306

3307

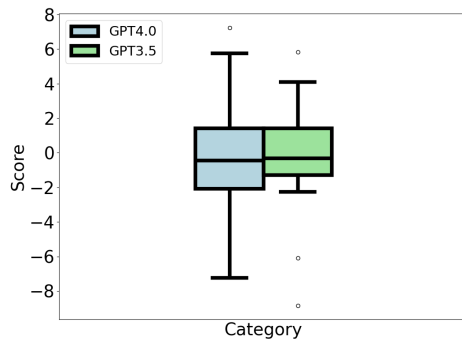
3308

3309

3310

3311

3312



3313 Figure 16: Game play performance of base LLM models on six card GOPS: GPT3.5 and GPT4.0.
3314 We see that the two language models perform similarly with our method.

3315

3316 K.0.2 SEARCH BUDGET

3317

3318 How does the effectiveness of the search+LLM agent scale with regards to the search budget? Does
3319 having a larger search process help achieve better performance?

3320

3321 Table 7: Average score difference for MCTS (num_rollout=32) + LLMFunction (Player1, top-3 func-
3322 tions shown in the table) vs. MCTS (num_rollout=32) + RandomRollout (Player2, num_rollout=10);
3323 100 games for each experiment;

3324

budget	Best Func.		2nd Best Func.		3rd Best Func.	
	Player 1	Player 2	Player 1	Player 2	Player 1	Player 2
16	-0.91	0.91	-0.7	0.7	-0.88	0.88
32	-0.95	0.95	0.44	-0.44	-0.73	0.73
64	-1.14	1.14	1.15	-1.15	0.46	-0.46
128	-1.28	1.28	0.36	-0.36	0.25	-0.25
256	-0.45	0.45	-0.85	0.85	-0.42	0.42
inf	-1.5	1.5	-2.26	2.26	-1.03	1.03

3333

3334

3335

3336

3337

3338

3339

3340

3341

3342

3343

3344

3345

3346

3347

3348
3349
3350
3351
3352
3353
3354
3355
3356
3357
3358
3359
3360
3361
3362
3363
3364
3365
3366
3367
3368
3369
3370
3371
3372
3373
3374
3375
3376
3377
3378
3379
3380
3381
3382
3383
3384
3385
3386
3387
3388
3389
3390
3391
3392
3393
3394
3395
3396
3397
3398
3399
3400
3401

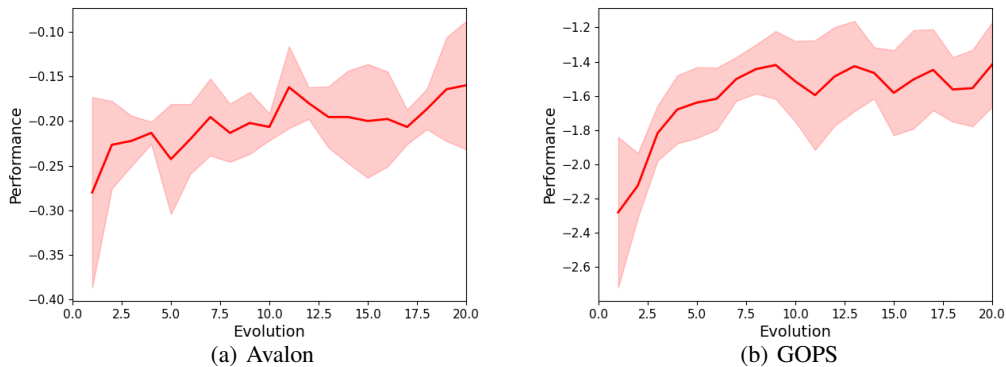


Figure 17: The training curves of the value heuristic via reinforcement learning in five-player Avalon and five-card GOPS, averaged across five independent runs. The solid lines show the mean and the shaded areas depict the standard deviation.

L DETAILS ON LEARNING THE VALUE HEURISTIC VIA REINFORCEMENT LEARNING

We employ Monte-Carlo based RL approach (Sutton & Barto (2018)) to train a value heuristic for both five-player Avalon and five-card GOPS games. To do so, we construct a MSE loss in each episode for training the value function, i.e.,

$$\operatorname{argmin}_{\theta} \sum_i^{\mathcal{N}} \sum_{t=0}^T \left(V_{\theta}^i(s_t) - \operatorname{Score}^i(s_t) \right)^2$$

where \mathcal{N} represents the number of actors, $V_{\theta}^i(s_t), i = 1, 2, \dots, \mathcal{N}$ denotes the value function for each actor, and T is the time horizon. Notice that s_t and $\operatorname{Score}^i(s_t)$ denote the state at time step t and the corresponding cumulative reward for each actor, i.e., $\sum_t^T R_i(s_t, a_t)$. It is worth pointing that $\operatorname{Score}^i(s_t)$ (the cumulative reward starting from s_t) is the unbiased estimate of the value function $V_{\theta}^i(s_t)$.

For both Avalon and GOPS games, the value function $V_{\theta}^i(s_t)$ is predicted by a neural network. We then train the value function network by minimizing the aforementioned loss function over episodes. In Avalon, we consider 20 evolutions (epochs) for the training process. At the end of each evolution, 30 batch runs (episodes) are generated and used to train the value function network, i.e., a total of 600 episodes for training. In GOPS, we train by 20 evolutions as well while considering 60 batch runs each (1200 episodes in total). We evaluate the final performance over 10 episodes in both games. The neural network is constructed by a multilayer perceptron (MLP) with 2 hidden layers. We select a hidden layer size of $128 * 128$ for Avalon and that of $64 * 64$ for GOPS. Likewise, the chosen learning rates are $5e - 4$ and $8e - 4$, respectively. The value function is expected to predict the score for each player in the game, e.g., two for GOPS and number of players for Avalon. All experimental hyper-parameters are summarized in Table 8.

Having introduced the set up, one can observe in Figure 17 an increased performance of RL-trained value heuristic in both five-player Avalon and five-card GOPS games. This validates the improvement for training value heuristic via reinforcement learning within limited evolutions.

3402
3403
3404
3405
3406
3407
3408
3409
3410
3411
3412
3413
3414
3415
3416
3417
3418
3419
3420
3421
3422
3423
3424
3425
3426
3427
3428
3429
3430
3431
3432
3433
3434
3435
3436
3437
3438
3439
3440
3441
3442
3443
3444
3445
3446
3447
3448
3449
3450
3451
3452
3453
3454
3455

Table 8: Summary of experimental hyper-parameters in RL-training value heuristic

Parameters	Avalon	GOPS
Type of neural network	MLP	MLP
Number of hidden layers	2	2
Hidden layer size	128*128	64*64
Learning rate	5e-4	8e-4
Output dimension	# of players	2
Number of evolutions	20	20
Number of batch runs	30	60
Number of final batch runs	10	10

M EXPERIMENTAL COMPUTE RESOURCES

All experiments in this work were performed on a workstation with an NVIDIA GeForce RTX 3070 GPU, Intel Core i9-10900 CPU at 2.80 GHz, and a Macbook Pro.

3456 N IMPROVEMENT METHOD BASELINES
3457

3458 **Line search** is a straightforward iterative process that continuously builds upon the most recent
3459 improved strategy. In each iteration, a strategy is generated and evaluated based on feedback from
3460 the environment. The feedback is then used to enhance the strategy via the LLM, and this cycle is
3461 repeated—similar to the Reflexion framework (Madaan et al., 2024). The essence of line search lies
3462 in its focus on immediate feedback, where only the latest strategy is considered for improvement,
3463 ensuring that progress is always aligned with the most recent understanding of the problem space.

3464 **Greedy search** takes a more competitive approach by selecting the best-performing strategy from
3465 the last generation to serve as the foundation for the next cycle of improvements. During each cycle,
3466 the best strategy is used to generate n variations, each an attempt at improvement. These variations
3467 are evaluated, and feedback from the environment is gathered to determine the best candidate for
3468 the subsequent cycle. This method is inspired by the Eureka framework (Ma et al., 2023), where
3469 LLMs are leveraged to iteratively refine reward functions in a manner akin to evolutionary algorithms,
3470 focusing on a winner-takes-all selection mechanism.

3471 **Best-first search** generalizes the improvement process by considering the top k strategies at each
3472 iteration, as opposed to focusing on a single best option. This is reminiscent of beam search but with
3473 a flexible branching factor that allows for the generation of multiple strategies from each selected
3474 candidate. By expanding multiple promising pathways simultaneously, the search can escape local
3475 optima and explore a broader solution space. This approach is related to the Tree of Thought
3476 method (Yao et al., 2024), which similarly employs a branching mechanism to explore various
3477 strategies in parallel during self-improvement cycles.

3478 **Best-first search with thought** enhances best-first search by incorporating a reasoning layer into
3479 the improvement process. Before generating new strategies, the LLM is prompted to first refine the
3480 underlying “thoughts” or decision-making processes that led to the generation of the top k strategies,
3481 including possible improvement ideas, before generating a new strategy. This meta-cognitive step
3482 draws from the React framework (Yao et al., 2022b), which emphasizes reasoning and actions during
3483 strategy development, adding an introspective element that encourages deeper exploration of strategy
3484 refinements.

3485 STRATEGIST introduces an additional layer of guidance through the use of an idea queue, Q . In
3486 this approach, ideas are generated and stored in Q , providing a reservoir of potential directions for
3487 improvement. The LLM uses these ideas to steer the strategy enhancement process, enabling more
3488 structured exploration. By separating idea generation from strategy improvement, STRATEGIST
3489 facilitates a more focused and deliberate search, ensuring that each iteration explores both immediate
3490 refinements and novel directions.

3491 All methods were run with the same computational budget—i.e., each method was allowed to
3492 generate the same number of improved strategies—ensuring a fair comparison across approaches in
3493 our experiments.

3494 We display a scaling curve for the various improvement methods in Figure 18.
3495
3496
3497
3498
3499
3500
3501
3502
3503
3504
3505
3506
3507
3508
3509

3510
 3511
 3512
 3513
 3514
 3515
 3516
 3517
 3518
 3519
 3520
 3521
 3522
 3523
 3524
 3525
 3526
 3527
 3528
 3529
 3530
 3531
 3532
 3533
 3534
 3535
 3536
 3537
 3538
 3539
 3540
 3541
 3542
 3543
 3544
 3545
 3546
 3547
 3548
 3549
 3550
 3551
 3552
 3553
 3554
 3555
 3556
 3557
 3558
 3559
 3560
 3561
 3562
 3563

Computation budget vs performance for different methods

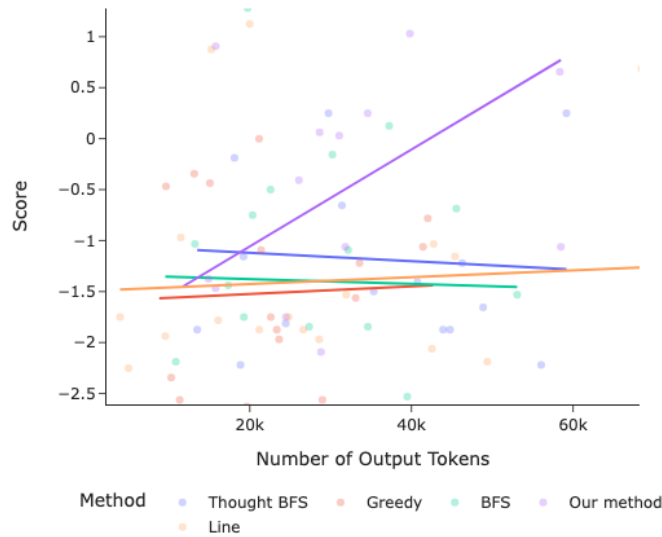


Figure 18: Number of output tokens from LLM vs game-play performance of generated value heuristics for 6-card GOPS. Each method was run 20 times at different token budgets, and the best function generated by each method was benchmarked against a baseline opponent policy.

O HUMAN EVALUATION DETAILS

We recruited ten experienced graduate students to play a total of fourty games of Resistance: Avalon with six players. Human participants were randomly assigned a character, while the other five characters were randomly assigned to STRATEGIST agents. The characters we played with were as follows: Merlin, 3 Servants of Arthur, Minion of Mordred, Assassin. Players were asked to complete a survey before the game and after the game. In the pre-game survey, players were asked to rate their own skill at Resistance: Avalon on a scale of 1 to 6. These results are shown in Figure 19 and Figure 20. We see that generally the players understand the rules well and are adept at Avalon.



Figure 19: **Familiarity** of human players with Resistance Avalon

Participants were asked to play one game of Avalon with dialogue against the STRATEGIST agents, which takes around 30-40 minutes per game, along with three games of Avalon without dialogue, which took around 10 minutes per game. After the experiment, participants were asked to rate the STRATEGIST agent on the seven key metrics presented in Figure 4 on a scale of 1 to 6. The experimenter was also asked to complete a survey rating the human participants on the same seven

3564
3565
3566
3567
3568
3569
3570
3571
3572
3573
3574
3575
3576
3577
3578
3579
3580
3581
3582
3583
3584
3585
3586
3587
3588
3589
3590
3591
3592
3593
3594
3595
3596
3597
3598
3599
3600
3601
3602
3603
3604
3605
3606
3607
3608
3609
3610
3611
3612
3613
3614
3615
3616
3617

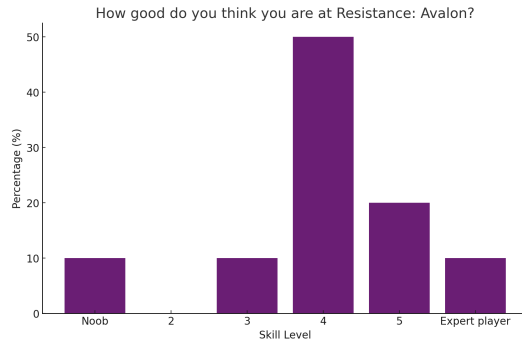


Figure 20: Skill level of human players at Resistance Avalon

key metrics. Participants were also asked to choose which word best described the STRATEGIST agents out of the following words: logical, deceptive, emotional, aggressive, cautious. Similarly, the experimenter was asked to choose which word best described the participant. These results are shown in Figure 5.

3618
3619
3620
3621
3622
3623
3624
3625
3626
3627
3628
3629
3630
3631
3632
3633
3634
3635
3636
3637
3638
3639
3640
3641
3642
3643
3644
3645
3646
3647
3648
3649
3650
3651
3652
3653
3654
3655
3656
3657
3658
3659
3660
3661
3662
3663
3664
3665
3666
3667
3668
3669
3670
3671

P RELATED WORKS DETAILS

In this section, we provide an in-depth discussion of related works that were briefly mentioned in the main paper:

- **ICL-AIL** (Fu et al., 2023) includes textual feedback and descriptions of previous self-play experiences as few-shot prompts to improve the negotiation abilities of the LLM-agent. The framework iteratively improves negotiation strategies by incorporating AI-generated feedback and historical dialogues.
- **SPAG** (Cheng et al., 2024) employs reinforcement learning to train an LLM-agent for the game Adversarial Taboo. The agent learns reasoning and adversarial strategy improvements through self-play, resulting in enhanced performance across various reasoning benchmarks.
- **Agent-Pro** (Zhang et al., 2024) updates textual memories (prior experiences) and textual beliefs about itself and the environment through reflection as it plays Blackjack and Texas Hold'em. The agent's policy optimization enhances its decision-making capabilities, outperforming baseline models.
- **LARLWorf** (Xu et al., 2023b) integrates reinforcement learning to optimize the action distribution suggested by the LLM. This approach addresses the biases in the LLM's suggestions and results in superior strategic play in the social deduction game Werewolf.
- **ComWorf** (Xu et al., 2023a) creates an LLM-agent that reflects on past experiences to improve its performance in the game Werewolf. The framework uses retrieval and reflection on prior interactions without requiring additional parameter tuning.
- **EnReaWolf** (Wu et al., 2024) leverages a dataset of human games to fine-tune an LLM-agent for the Werewolf game. This fine-tuning improves both reasoning and communication capabilities, allowing the agent to surpass standard LLM performance.