STRATEGIST: Learning Strategic Skills by LLMs via Bi-Level Tree Search

Anonymous Author(s)
Affiliation
Address
email

Abstract

In this paper, we propose a new method STRATEGIST that utilizes LLMs to acquire new skills for playing multi-agent games through a self-improvement process. Our method gathers quality feedback through self-play simulations with Monte Carlo tree search and LLM-based reflection, which can then be used to learn high-level strategic skills such as how to evaluate states that guide the low-level execution. We showcase how our method can be used in both action planning and dialogue generation in the context of games, achieving good performance on both tasks. Specifically, we demonstrate that our method can help train agents with better performance than both traditional reinforcement learning-based approaches and other LLM-based skill learning approaches in games including the Game of Pure Strategy (GOPS) and The Resistance: Avalon.

1 Introduction

Recent studies have demonstrated how Large Language Models (LLMs) can be utilized to learn skills for improved decision-making in interactive environments [1, 2]. However, learning skills in adversarial environments with multiple agents presents a significant challenge for LLMs, as it requires accounting for the responses of other players or environment to their actions. In these environments, it is hard to determine the optimal policy, since strategies that are effective against one opponent's policy may not work for another, and the opponents can adapt to our current strategy. Moreover, the complexity of these environments means that we need to smartly search across a large policy space to find the optimal policy. Hence, existing LLM self-improvement methods that rely only on feedback from the LLM often have trouble distinguishing what improvements help and end up in local optima strategies that can be exploited.

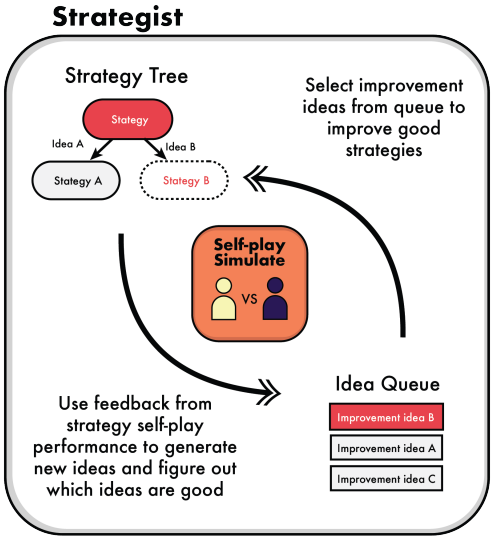


Figure 1: **Overview of STRATEGIST**. Our method learns high-level skills/strategies through figuring out how to search and improve previously learned strategies. We use low-level self-play simulations to guide the high-level search process.

In this paper, we study how to (1) acquire and analyze feedback in these noisy environments and (2) make sure that the improvements actually help against different opponents. We propose a method, referred to as STRATEGIST, that learns an effective policy, evaluates the policy in the absence

Submitted to 38th Conference on Neural Information Processing Systems (NeurIPS 2024). Do not distribute.

of the real environment at a low level, and finds such a policy efficiently. In other words, we are also *learning how to learn!*

Specifically, our approach combines LLM self-improvement with search, both on a high level strategy (or skill) space, and also on a low-level action space. On the **high-level** we do this by constructing a strategy tree in an evolutionary process, where we improve upon previously generated strategies. In our game-play settings, we specifically learn a value heuristic to evaluate game states, and a dialogue guide to guide the dialogue generation process. During the process, we also learn how to improve strategies, and what types of improvements tend to be more helpful generally. We search across the strategy tree using exploration and exploitation methods, eventually finding good strategies. No parameter updates or supervised fine-tuning are necessary during the search process.

The strategies are tested out through simulated self-play on the **low level**. By mastering a high-level strategy, the LLM not only learns a good policy for the player, but also gains insights into how opponents might react to that policy. With this information, our method can simulate the behaviors of both players and opponents by MCTS, gathering feedback through the simulated trajectories and the search process. In other words, *we leverage the world-knowledge in the LLM to simulate behaviors of both players and opponents in order to get better feedback and plan better.*

We apply our method to two games – Game of Pure Strategy (GOPS) and Resistance: Avalon – and show that our improvement process works for both games with dialogue and games without. We find that our improvement method is better than many other self-improvement methods, including both existing LLM-based self-improvement methods and traditional RL-based training approaches to AI in games, given the same computation and data budget. We also show the effectiveness of our method of collecting simulated self-play feedback, including using search to provide feedback.

To summarize, our main contributions are as follows:

- We propose a general non-parametric LLM skill-learning framework, in which a low-level self-play tree-search provides value feedback, and a high-level skill learning module improves the skill in various forms, including functions and textual strategies.
- To improve the sample efficiency of skill learning, we introduce a modular search method, which factorizes the process by an idea-implementation search with bandit exploration.
- We apply this framework to GOPS and Avalon, and show that through learning a value heuristic and dialogue strategy guide we can learn a good policy for both action planning and dialogue generation. Some examples of good strategies learned through our method are shown in G.3 and H.3, which achieve higher win-rates against other agents.

2 Methodology

2.1 Strategy Learning in Decision Making Setting

The general framework for STRATEGIST is shown in Figure 2 with pseudo-code 1. Our improvement process contains two improvement steps in each improvement cycle – the (1) reflection and idea generation step and (2) the strategy improvement step. During the idea generation step we prompt the LLM to reflect on simulated self-play feedback from previously evaluated strategies and generate possible improvement ideas to the strategies and add them to the idea queue. During the strategy improvement step, we select a strategy from the strategy tree and an improvement idea from the idea queue and prompt the LLM to improve the strategy using the improvement idea. The improved strategy is then evaluated via self-play simulations, and we use the feedback and reward signals from the simulation to help guide future improvements.

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)), which can be done by improving strategies associated with the policy function. We describe in more detail what a strategy looks like, how we derive a policy function from a strategy, and how to acquire feedback for the strategy for both dialogue generation and action generation here.

Problem definition. Given state space $\mathcal{S}$ and action space $\mathcal{A}$, a policy function ϕ in policy space Φ is a mapping $\phi : \mathcal{S} \to \Delta\mathcal{A}$ 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 $T : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$, a reward function $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^{|\mathcal{N}|}$ that specifies

intermediate rewards for each actor, an action function $A : \mathcal{S} \to \mathcal{N}, \mathcal{P}(\mathcal{A})$ that specifies which actor may take what legal actions at some state where $\mathcal{P}$ is power set, and ϕ_e, the policy function for the environment actor. Note that transitions are deterministic in our notation, and stochastic transitions are handled by the environment actor $e \in \mathcal{N}$ 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 $H : \mathcal{S} \times \mathcal{N} \to \mathcal{I}$ that maps from hidden states and actors to hidden information sets. Hence, $\phi : \mathcal{I} \to \Delta\mathcal{A}$ is a function from information sets to action distributions instead.

Let $f : \Sigma \to \Phi$ 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. 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.

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

where $\tau = (s_0, a_0, ...)$ is the simulated trajectory according to the joint policy (ϕ_i, ϕ_{-i}) and the transition function T, with $a_t \sim \phi(a_t|s_t)$ and $s_{t+1} = T(s_t, a_t)$.

The state space, action space, and actor space are different depending on the setting. In non-stochastic, single agent settings such as question answering [3], $\mathcal{N} = \{0\}$. In stochastic single agent settings such as WebShop [4], $\mathcal{N} = \{-1, 0\}$ where we add an environment agent -1. We specifically focus on adversarial, multi-agent stochastic game ($\mathcal{N} = \{-1, 0, 1, ...\}$) 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, $\mathcal{N} = \{-1, 0, 1\}$, 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 (which we will call moves) such as voting, and also all possible natural language texts of a given length. Similarly, $\mathcal{S}$ consists of both the historical moves and historical dialogue record, and $|\mathcal{N}| \geq 6$.

Skills (strategies) here refer to any high-level strategy space Σ where strategies σ can be learned. In our Avalon game setting for example, we learn (1) a value heuristic $\sigma := v : \mathcal{S} \to \mathbb{R}^{|\mathcal{N}|}$ (value function, see Appendix G.3 for example), a strategy to evaluate different states of the game (served as world model) and (2) a textual strategy guide on how to evaluate and analyze the current discussion and the steps for generating good dialogue (see Appendix H.3 for example).

We focus on learning high-level skills because this allows for higher level **strategic planning and LLM reasoning**, abstracting away low-level details that may be scenario specific and thus generalizing better to different cases. Given the size of the search space, it is also computationally expensive to search for the best action for each input state. While the skill coach helps us learn high-level strategies, the evaluator executes and evaluates the high-level strategies on the lower level. In our setting, this is usually done through **game simulations and self-play**, where the evaluator compares agents who use different strategies against each other.

2.2 STRATEGIST for High-Level Strategy Learning

Our main methodological contribution lies in the development of a self-improvement method STRATEGIST that utilizes LLMs to learn new skills *without direct supervision, where feedback comes from simulated self-play*. We further use modular search to improve the sample efficiency. The method generalizes to different settings, and we show different ways to use our method in the next sections.

The framework behind the STRATEGIST is shown in Figure 2, with pseudo-code 1. The skill coach maintains a strategy library of the strategies it has generated so far, starting from some seed strategies, along with the performance score of the strategy and raw feedback on how the strategy performed in practice. It also maintains a queue of improvement ideas, ways that we can improve strategies, along with how much the idea improves strategies on average (score). We keep track of which strategies evolved from which and what improvement idea it used, which forms a tree structure. There are two alternating improvement loops that we run – the **reflection and idea generation step**, and the **strategy improvement step**. We run these two loops for a fixed number of improvement cycles. When improving, we use an **adaptive selection policy** such as upper confidence bound (UCB) or best first search (BFS) to select strategies and ideas to improve upon.

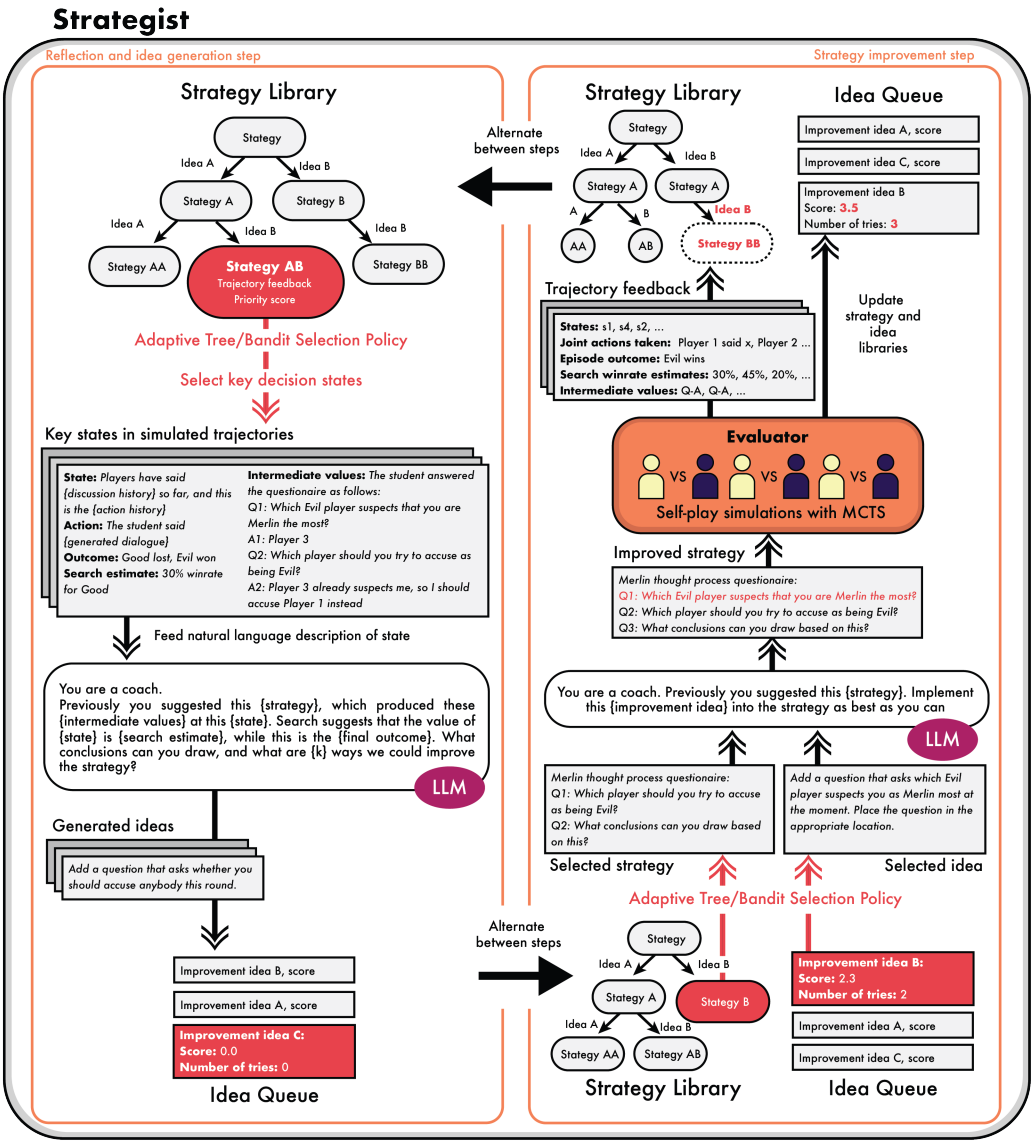


Figure 2: **Overview of our skill coach improvement method**. There are two improvement steps – the idea generation step and the strategy improvement step and we alternate between the two.

During the **idea generation step**, a strategy σ and its simulated trajectory feedback τ_σ is first selected from the strategy tree according to the adaptive selection policy. The feedback usually trajectories from previous self-play simulations, including the states visited, actions taken at those states, the estimated win-rate at that state, the final outcome of the trajectory, and any intermediate values used during simulations at those states. Since these trajectories are very long (around 30 steps each for Avalon), we select some key states in the trajectory to translate into natural language. We then prompt the LLM to reflect on this feedback and propose a couple of new ideas on how to improve the function. These new ideas are added to the idea queue, along with a score prior to how good the idea is.

During the **strategy improvement step**, we first select a strategy and an idea from the libraries using the adaptive selection policy. We then prompt the LLM to implement the idea of the strategy, generating a new improved strategy. We then evaluate the strategy using the evaluator, which conducts self-play simulations with the strategy, and records the simulated trajectory data. During simulations, players conduct an MCTS tree search to estimate the expected win-rate at different states, which provides additional feedback. We add the improved strategy (and its performance) to the strategy tree, and update the improvement score for the idea that was used.

A key challenge during the improvement process is that it is often difficult to evaluate whether an improved strategy is better than the old one and why. Simply querying the LLM to improve the strategy based on the feedback without guidance from an improvement idea often results in the LLM making changes to all parts of the strategy. Thus, after we evaluate the strategy, it is hard to determine what specific change improved it. Hence, we utilize an additional idea queue to **modularize the search process**. Instead of improving everything all at once, we *incrementally improve the strategy, ceteris paribus*. This creates a more scientific testing and improvement process that avoids confounding factors.

Moreover, the idea queue helps us keep track of what improvements have been successful in the past, which are also likely to help in the future. UCB (upper confidence bound) sampling from the idea library helps us both explore new improvement ideas, and exploit old ones. Our UCB equation is

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

where $\overline{z}$ is the empirical average improvement score, N_{total} is the total number of idea implementations, and $N_{\text{improvement idea}}$ is the number of times we have implemented this improvement idea specifically.

To make the improvement process more manageable, we **optimize the policy for dialogue generation and moves separately** in Avalon. We then integrate the two parts (refer to Appendix D), which together produces an agent fully capable of playing games with discussion. This type of integration of separate components has seen success in other settings such as Diplomacy [5].

2.3 Self-improvement Feedback for Non-dialogue Actions

For non-dialogue actions, while the action spaces $\mathcal{A}$ and state spaces $\mathcal{S}$ themselves are usually discrete and finite, the number of possible functions Φ from state space to action space is very large. Most LLM-agents query the LLM directly with the state information for next actions in decision-making environments [6, 7, 8]. However, we found this method to be costly since the LLM needs be queried for every move, and a game of Avalon usually has at least 20 moves per player. This becomes even costlier when we use any time of look-ahead search to improve the planning capabilities of the agent and does also needs to query the LLM for every look-ahead future actions and states. Traditionally reinforcement learning tackles the problem of having a large policy space by parameterizing the policy and optimizing the parameters instead, starting from a random initialization of the parameters, thus reducing the search space. Building on this approach, instead of querying the LLM every time for an action, we instead **parameterize the policy as a python function**, and we have the LLM search over the space of such functions instead.

With its inherent **world model**, the LLM can help us search and optimize over the policy space more effectively. Given the rules of the game in natural language form that a human can understand, the LLM can quickly generate policies that seem reasonable. In natural language form, it is often easier to describe the value of a state versus describing the optimal action to take in a given state. Humans often mentally go through the process of comparing different states (outcomes) when making decisions. Thus, we opt to learn a value function, i.e. $\sigma := v : \mathcal{S} \to \mathbb{R}^{|\mathcal{N}|}$, an approach that is often used in reinforcement learning to help stabilize the training process. It is easy to convert from a value function to a policy function since we can simple take the action that leads to the best state, i.e. $\phi_i(s) = \text{argmax}_{a \in \mathcal{A}} Q(s, a) = \text{argmax}_{a \in \mathcal{A}} R_i(s, a) + v_i(s') | s' = T(s, a)$.

We call it a **value heuristics function** because, given the little data that we improve on, it is an inaccurate estimate of the true value function. Hence, to resolve these inaccuracies, we additionally enhance the policy function with MCTS, used commonly in other AI agents such as Alpha-go and Muzero [9, 10]. The MCTS process is depicted in Figure 10. We provide more details on our MCTS implementation in the Appendix E.

The **search process** allows us to get better value estimates than the initial ones given by the value heuristic when making decisions. More importantly, this search process also allows us to generate more feedback than normally possible, since we can compare the updated value estimate computed through MCTS with the initial estimate from the value heuristic. *The estimated win-rate from search also provides a shaped process reward, which is better than the outcome reward of either winning or losing as a reward signal.*

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
```

2.4 Self-improvement Feedback for Dialogue Generation

Dialogue generation presents another distinct challenge for self-improvement. In dialogue generation, both the action spac $\mathcal{A}$, i.e. the number of possible sequences of words one could generate for any discussion round, the state space $\mathcal{S}$, i.e. the number of possible input dialogue from previous rounds, are huge. This means that the number of possible dialogue generation policies Φ is massive, and traditional parameter training approaches have great difficulty optimizing across this space. We solve this problem by learning a high-level **strategy guide** that the agent uses when speaking. Specifically, the strategy guide formalizes a process to think about how to generate dialogue, given the situation. This is implemented as a question and answer worksheet. The agent is first instructed to answer all the questions in the strategy guide before using it as a prompt to generate dialogue.

Moreover, it is difficult to provide good **reward signals** during the dialogue training process. The first problem is the lack of available training data. Existing dialogue generation methods for games often rely on imitation learning on existing human generated text data through parameter training of the underlying network [5]. However, in many settings, such large quantities of human generated data might not be readily available for the tuning of language models. The second problem lies in how to accurately evaluate how the generated dialogue performs. Players need to **optimize and balance multiple objectives** when discussing. For example, in Avalon, the Merlin player needs to both hide themselves from Evil and convince the Good players that they are Good through discussion.

Example 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 remain vigilant and not allow ourselves to be swayed by potentially deceitful tactics."

We solve these problems by first simulating a game using some initial policy ϕ_0 and storing the simulation in a scenario database. Then during evaluation, we sample scenarios from the database to evaluate with. A **scenario** is a state in the game when a player had to take a dialogue action. We give the player a history of the previous discussions and actions up to the decision point as if it played the game up to that point, and ask it to generate new dialogue using the new strategy guide. Then we simulate the thought process of the opponents as if they also played the game up to that point, and we get their belief updates. Using the belief updates we then score the performance of the player that uses the improved strategy guide, and provide the opponent thought process as feedback. We provide more details on our implementation in Appendix F.

3 Experiments

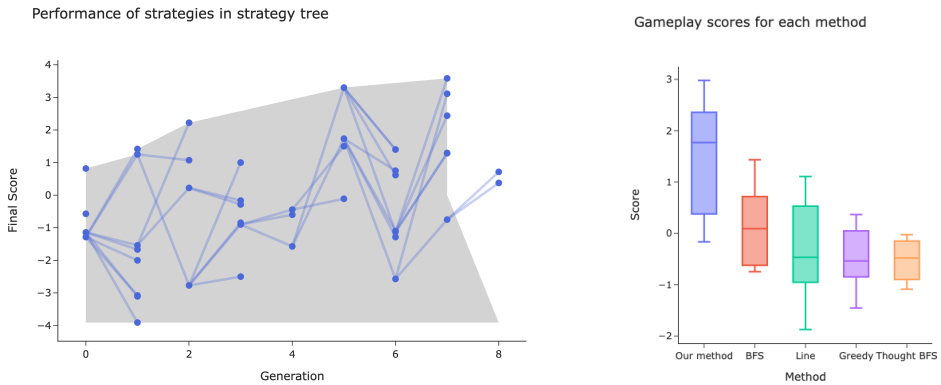


Figure 3: **Left**: Example performance of value heuristics strategy tree for GOPS. Points represent the final evaluation score and generation of improved functions. Lines depict which function the improved function evolved from. Our method is able to escape local maxima and continue exploring the strategy space. Note that these are final game-play scores against a fixed distribution of opponents, which differs from the intermediate self-play scores used to select the best strategy to improve upon during the improvement process. **Right**: Comparison of different improvement methods on 6 card GOPS, where we play 9 functions generated by each method against each other for 1024 games total.

We demonstrate the effectiveness of our self-improvement process through experiments against different improvement benchmarks. We tested our method on (1) GOPS, a two-player zero-sum card game (see B for rules) and (2) Avalon, a five or more player team-based discussion game (see A for rules). For Avalon dialogue generation, we specifically benchmark on the Merlin role, since that is the hardest role to play. Details on how we implemented policies from learned strategies and acquired feedback are described in Section 2.1, 2.3, and 2.4, and Appendix D, E, and F.

3.1 Different LLM Improvement Methods

We demonstrate the effectiveness of our strategy improvement method by benchmarking it against four other skill-improvement methods. **Line search** [11] always reflects and improves upon the latest improved strategy. **Greedy search** [12] selects the best strategy from the last generation of improved strategies to improve upon each improvement cycle. **Best first search** [13] improves upon the k best strategies generated in any iteration of each improvement cycle. **Best first search with thought** asks the LLM to improve upon the thoughts used to generate the k best strategies before improving the strategy itself. STRATEGIST is our method that uses an additional idea queue Q and an idea generation step to guide the improvement process.

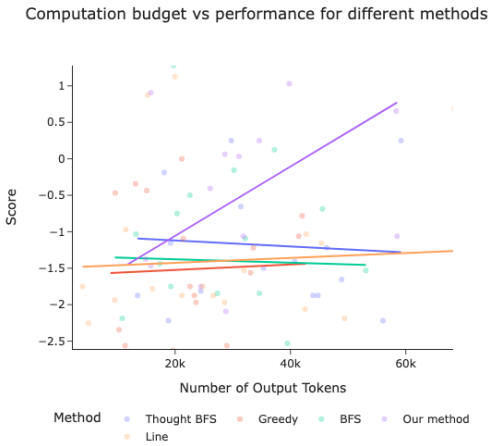


Figure 4: Number of output tokens from LLM vs game-play performance of generated value heuristics for 6-card GOPS. Each method was run 20 times at different token budgets, and the best function generated by each method was benchmarked against a baseline opponent policy.

Our results are shown in Table 1, where the method of collecting feedback (simulational self-play) is the same but we vary the improvement method. The number of new strategies generated by each method is also held constant. The **gameplay scores** of playing the strategies generated by each method against each other on GOPS is shown in Figure 3 right. Even when we control for the number of output tokens generated by the LLM, we see that our method still achieves higher performance as shown in Figure 4. We believe that the reason why we see higher performance in our method is because (1) the idea queue helps test which incremental improvements are helpful and

guide the search process and (2) our strategy and idea selection policy help us explore the strategy space more efficiently and escape local maxima as shown in Figure 3 left.

Table 1: **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. The median scores ($\pm$ IQR/2) for each method are shown. 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 F.

Self-Improvement Methodology	**Line search** [11, 7]	**Greedy search** [12]	**Best First Search (BFS)** [13]	**BFS with thought**	**STRATEGIST**
GOPS Value Heuristic	-0.47 $\pm$0.74	-0.54 $\pm$0.45	0.092 $\pm$0.67	-0.48$\pm$0.375	**1.5** $\pm$0.99
Avalon Value Heuristic	0.54 $\pm$0.11	0.47 $\pm$0.11	0.50 $\pm$0.085	0.55 $\pm$0.065	**0.59** $\pm$0.11
Avalon Dialogue Guide	0.37 ± 0.19	0.62 ± 0.13	0.49 ± 0.063	0.37 ± 0.06	**0.88** $\pm$0.063

3.2 LLM-improvement vs Reinforcement Learning (RL) Training

We demonstrate the effectiveness of our method against traditional RL-based approaches to learning a good policy. Specifically, we show that our method is able to learn a value heuristic function more efficiently than deep RL, the approach taken by AlphaGo and MuZero [9, 10]. We know that given enough training data, training time, and a large enough network, deep RL can get arbitrarily close to the true value function for the game. To make the comparison between RL and LLM fair, we (1) limit RL and the LLM to seeing the same number of simulated episodes (2) limit the number of training steps after seeing a batch of episode trajectory data. Note that in actuality, the LLM will only be seeing a very small sample of the data ($N = 10$ steps out of the entire batch of episodes), while the RL algorithm sees all of the simulated data. Our results are shown in Table 2 and Figure 5, where we see that STRATEGIST outperforms RL consistently in both settings. We also benchmark against a **random rollout** value heuristic where you simulate (rollout) the game using a random policy and take the average cumulative rollout reward as the value, commonly used in AI agents [9].

Table 2: **Comparison of reinforcement learning vs our method**. We run each process 10 times, taking the best strategy generated by each run and playing them against each other for 1024 games.

Setting / Improvement method	**Random Rollout**	**RL-Training**	**STRATEGIST**
GOPS Value Heuristic	0.72 ± 0.00	0.31 ± 0.35	0.66 ± 0.60
Avalon Value Heuristic	0.23 ± 0.00	0.30 ± 0.11	0.38 ± 0.12

3.3 Feedback Quality and Reward Signal

Recent works tend to focus on either using another LLM to critique the generations and thus provide feedback [11, 7], observations from real environment interactions [14], or some combination of the two [1]. Since our method learns how to internally simulate the opponents by learning a strategy and thus gain insights into the opponent's policy, we are able to acquire better quality feedback than either approach. We benchmark our feedback acquisition method against (1) using a **LLM-critic** and (2) trajectory feedback from interactions against a **fixed opponent policy**. Our results are shown in table 3, where our method achieves better performance on both action planning in GOPS and dialogue generation.

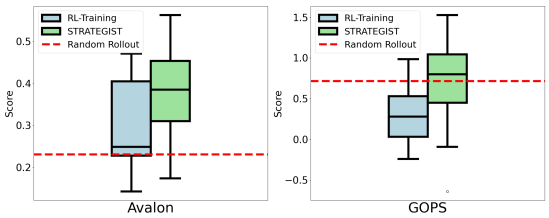


Figure 5: Performance of different training methods against the baseline in Avalon and GOPS.

4 Related Work

LLMs for text agents. Large language models (LLMs) have demonstrated significant emergent capabilities, such as zero-shot prompting and complex reasoning [15, 16, 17, 18, 19, 20]. They

Table 3: **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.

Setting/Method	LLM-critic [11, 7]	Environment feedback with fixed opponent [14]	Simulated Self-play Feedback [STRATEGIST]
GOPS Value Heuristic	-0.27 ± 1.1	0.089 ± 0.86	**0.87** ± 1.5
Avalon Dialogue Guide	0.37 ± 0.063	0.62 ± 0.13	**0.88** ± 0.063

also possess extensive world knowledge [21], which has spurred increasing efforts to use LLMs for decision-making in text agents [22]. One notable paradigm is ReAct [6], which employs an observation-reasoning-acting loop for agent planning with LLMs. Building on ReAct, Reflexion [7] incorporates self-reflection to enhance reasoning capabilities. Other works in this domain have utilized feedback [1, 23], memory [24], and tool use [25, 26] to further enhance agent performance. Our proposed method, STRATEGIST, integrates these components to design an agent capable of systematic analysis and strategic decision-making. Typical prompting techniques for text agents include Chain-of-Thought [27], Tree-of-Thought [13], and Graph-of-Thought [28]. While these techniques are effective for high-level reasoning, they are insufficient for complex games that require feedback signals for self-improvement. STRATEGIST adopts a bi-level tree search approach, enabling both high-level planning and low-level agent self-play for providing feedback.

Skill learning with LLMs. Recent works have explored the possibly of LLMs learning skills through learning a textual short and long term memory [7, 29], or textual insights extracted from the memories [8]. Due to the length of trajectories in our game setting and the numerical nature of the data, it is difficult to learn textual memories, so we learn high level strategies instead. We also explore how to acquire simulational self-play feedback in multiagent settings. Using LLMs to learn a functional reward model has also been applied to great success on single-agent robotic tasks [12, 30]. We build upon their work by introducing a new improvement method that can help learn a better reward model, and exploring how function learing can be applied to multiagent settings with simulated feedback.

AI in strategy games. AI has been applied to great success in board games. AlphaGo and MuZero demonstrated the power of combining MCTS, deep learning, and feedback generation using self-play in games such as Go, Chess, and Shogi [9, 10]. Language models can also be trained on human in-game discussion data and integrated with another separately trained action planner to play board games with dialogue [5]. We build upon the AI for games literature by showing that LLMs can accomplish both *(1) the training of a value heuristic like that in AlphaGo through self-play more efficiently than RL and (2) dialogue generation in discussion games with no human examples*. These adversarial environments are not just limited to board games. For example, there has been recent interest on creating LLM-agents that can negotiate [31, 32], which our method can also be applied to.

5 Limitations and Conclusion

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.

In conclusion, we have presented STRATEGIST, a generalizable non-parametric self-improvement framework that learns and improves skills. Given the rules of the game, our method is able to learn good strategies to play the game through self-play *without task-specific prompting or human generated policy data*. The performance of STRATEGIST suggests that incorporating better guidance, whether this be through modular high-level search or low-level simulated self-play feedback, into LLM-improvement processes can greatly enhance the improvement process.

References

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

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

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

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

[5] 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.

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

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

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

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

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

[11] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36, 2024.

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

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

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

[15] 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, Avanika 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.

[16] 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, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[17] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21:140:1–140:67, 2020.

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

[19] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv: Arxiv-2204.02311*, 2022.

[20] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. Scaling instruction-finetuned language models. *arXiv preprint arXiv: Arxiv-2210.11416*, 2022.

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

[22] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):1–26, 2024.

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

[24] Joon Sung Park, Joseph C O'Brien, Carrie J Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. *arXiv preprint arXiv:2304.03442*, 2023.

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

[26] Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. Large language models as tool makers. *arXiv preprint arXiv:2305.17126*, 2023.

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

[28] Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, et al. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 17682–17690, 2024.

[29] Bodhisattwa Prasad Majumder, Bhavana Dalvi Mishra, Peter Jansen, Oyvind Tafjord, Niket Tandon, Li Zhang, Chris Callison-Burch, and Peter Clark. Clin: A continually learning language agent for rapid task adaptation and generalization. *arXiv preprint arXiv:2310.10134*, 2023.

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

[31] 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.

[32] 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.

[33] Wikipedia contributors. Goofspiel — Wikipedia, the free encyclopedia, 2023. [Online; accessed 22-May-2024].

[34] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

A Resistance: Avalon Game Description

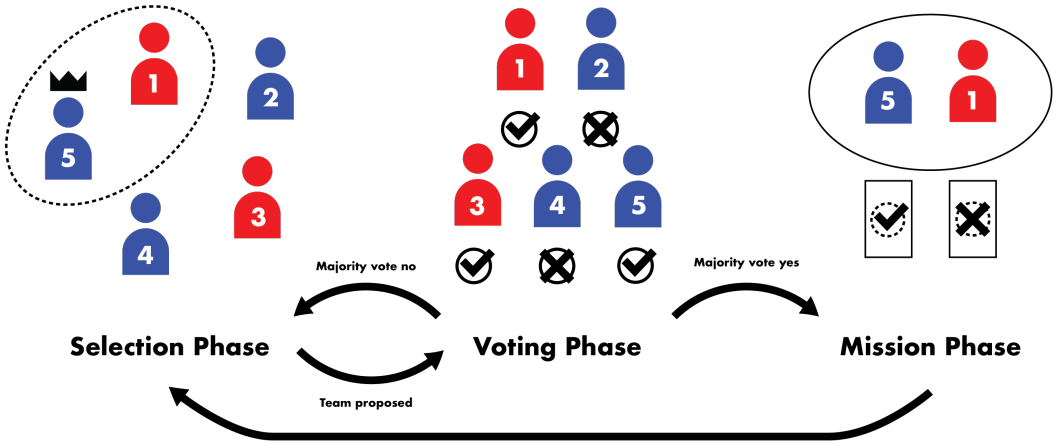


Figure 6: **The three phases per round of Resistance game**. Good players are shown in blue, while Evil players in red. In *Selection Phase*, the team leader (player 5 in this round) proposes a team (player 1 and 5, himself). In *Voting Phase*, all players vote publicly whether to approve this team or 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 8, 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 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

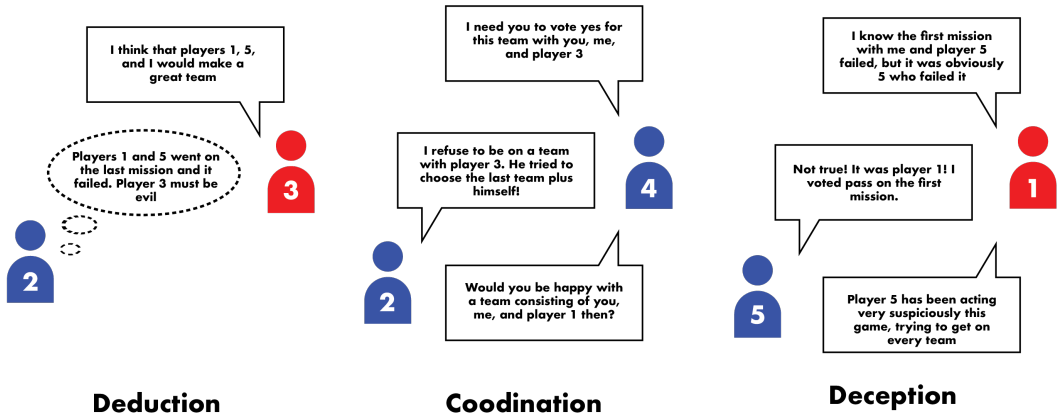


Figure 7: **Communication Skills required to play Avalon**. 1) First, they use logical reasoning to analyze the voting pattern and dialogue of other players and deduce their motives. 2) they must coordinate, communicate, and persuade their teammates to follow a particular strategy. 3) they must also hide their identity and motives through deception.

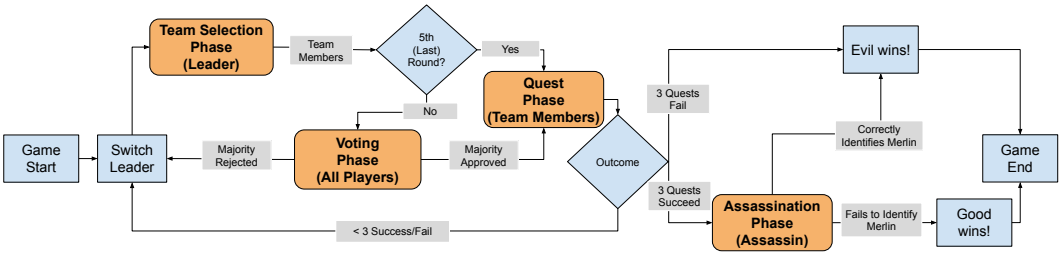


Figure 8: **Flowchart illustrating the various game states and transition diagram**. Round boxes indicate game states (phases) where the player (role highlighted in bracket) has to make decisions

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

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

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

A.4 Game Ending and Assassination

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

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

[1] At 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

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 themself 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.

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 ([33]). In our experiments we play 5 or 6 card GOPS. Specifically, the score cards are $\{1, 2, ...n\}$ and each player starts with a hand of cards $\{1, 2, ...n\}$ 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.

C Improvement Process Implementation Details

Algorithm 1: STRATEGIST Pseudocode

Data : T: strategy tree storing strategy s, feedback (τ_s), and priority score (z_s), Q: idea queue, 'seed functions', N_{ideas}: number of ideas, $N_{strategies}$: number of strategies, $N_{evolutions}$: number of evolutions, $N_{feedback_examples}$: number of states of give as feedback,

Function `select_strategy(`T`):`

  $\sigma_{\text{best}} \leftarrow \arg\limits_{\sigma \in T_2} \text{softmax} z_\sigma$ `// one possible implementation where you take one of the best two strategies in the whole tree randomly (BFS2)`

  return s_{best}

Function `select_idea(`Q, σ`):`

  $d_{\text{best}} \leftarrow \text{softargmax}_{d \in Q} UCB(z_d, n_d)$ `// one possible implementation where you take the best strategy in the queue using softmax UCB,` z_d `being the empirical` q`-value and` n_d `being the number of tries`

  return σ_{best}

Function `select_key_states(`τ_σ`):`

  $K_\sigma \leftarrow \arg\limits_{s \in \tau} \max_k (\text{SearchEstimate}(s) - v_\sigma(s))^2$ `// one possible way to select key states for` σ `that is a value heuristic` v_σ

  return K_σ

Function `generate_ideas(`N_{ideas}`):`

  $\sigma \leftarrow$ `select_strategy(`T`);`

  $K_\sigma \leftarrow$ `select_key_states(`τ_σ`)` `//` K_σ `is a set of key states from the trajectory feedback` τ_σ `for strategy` σ

  $D_{\text{new ideas}} \leftarrow$ LLM(Generate N_{ideas} new ideas based on string description of K_σ, which includes the output of the strategy, action taken, state description, final outcome of the trajectory, search estimate of the state, and any intermediate values used to compute the output of the strategy);

  for $d \in D_{new\ ideas}$ **do**

    Store d in Q with prior score $z_d = 0.0$ and $n_d = 0$;

  end

Function `implement_strategies(`$N_{strategies}$`):`

  $\Sigma_{\text{new}}, D, P =$ `[], {}, {}` `// list of new generated strategies, dictionary mapping new generated strategy to the idea that generated it, and dictionary mapping generated strategies to their parents`

  for $i \leftarrow 1$ **to** $N_{strategies}$ **do**

    $\sigma \leftarrow$ `select_strategy(`T`);`

    $d \leftarrow$ `select_idea(`Q, σ`);`

    $\sigma_{\text{new}} \leftarrow$ LLM(Improve σ using d);

    Σ_{new}.append(σ_{new});

    $D[\sigma_{\text{new}}] = d$;

    $P[\sigma_{\text{new}}] = \sigma$;

  end

  $W, \mathcal{T} \leftarrow$ SelfplaySimulate($\Sigma_{\text{new}} \cup \text{unique}(P.\text{values})$) `// simulate games, getting average winrates` $W[\sigma]$ `for each strategy` σ `and simulated trajectory feedback` $\mathcal{T}[\sigma]$

  for $\sigma \in \Sigma_{new}$ **do**

    T.add($\sigma, P[\sigma], D[\sigma]$) `// add new strategy to tree from parent based on idea`

    $z_\sigma \leftarrow W[\sigma]$ `// add function score`

    $z_{D[\sigma]} \leftarrow \frac{n_{D[\sigma]}}{n_{D[\sigma]}+1} z_{D[\sigma]} + \frac{1}{n_{D[\sigma]}+1}(W[\sigma] - W[P[\sigma]])$ `// update idea score with how much it improved the strategy by`

  end

repeat

  `generate_ideas(`N_{ideas}`);`

  `implement_strategies(`$N_{strategies}$`);`

until $N_{evolutions}$;

return *Best strategies in* T *according to their scores* z_s

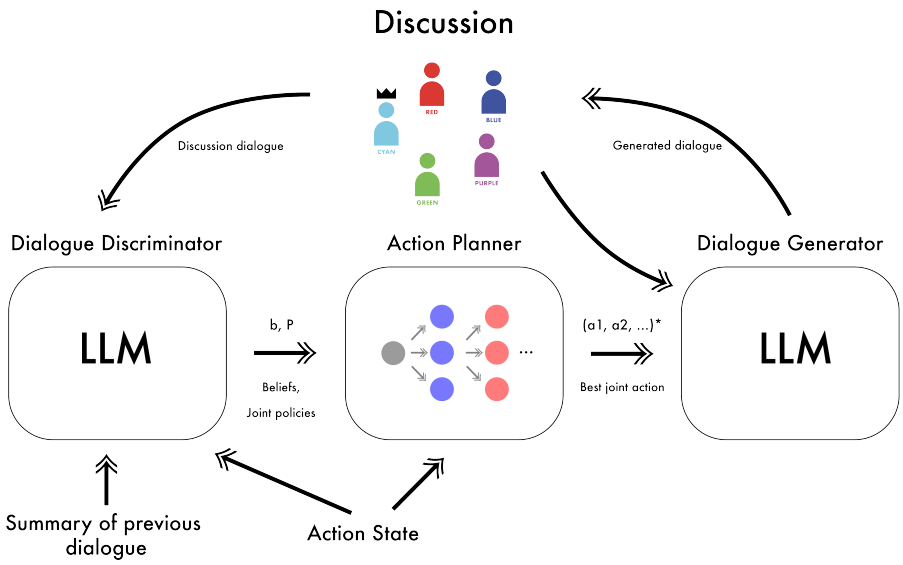


Figure 9: Overview of the LLM-powered agent, including the three main modules that we use to generate dialogue during discussion

D Avalon Agent Implementation Details

We describe in detail how we implement our model below and as shown in figure 9. 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.

D.1 Dialogue Analyzer (Discriminator)

The dialogue analyzer f_{ana} takes as input $\boldsymbol{I}$ information set (partial information) of the current state for the player, $\boldsymbol{d}_t$ the discussion so far this round, and $\boldsymbol{b}$ some prior beliefs about the hidden state of the game, and returns $\widehat{\boldsymbol{b}}$, the updated beliefs, and $\widehat{\boldsymbol{\Pi}}_t$, 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 $\boldsymbol{s}$ given the information set $\boldsymbol{I}$ using the dialogue analyzer.

$$\widehat{\boldsymbol{b}}, \widehat{\boldsymbol{\Pi}}_t = f_{ana}(\boldsymbol{I}, \boldsymbol{d}_t, \boldsymbol{b})$$

In the context of Avalon, $\boldsymbol{I}$ 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. $\boldsymbol{b}$ will contain information on (1) the probability of each player being Evil and (2)

the probability of each player being Merlin, both conditioned on the private information contained in $\boldsymbol{I}$. 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 $\boldsymbol{I}$, $\boldsymbol{d}$, $\boldsymbol{b}$ (converted to natural language form) as prompts, along with some instruction prompt $\boldsymbol{\phi}_{ana}$ that prompts it to produce $\widehat{\boldsymbol{b}}, \widehat{\boldsymbol{\Pi}}_t$. Specifically,

$$f_{ana}(\boldsymbol{I}, \boldsymbol{d}_t, \boldsymbol{b}) = f_{LLM}(\boldsymbol{\phi}_{dis}, \boldsymbol{I}, \boldsymbol{d}, \boldsymbol{b})$$

We show examples of such prompts in Appendix I.

D.2 Action Planner

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

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

More specifically, in our search implementation, at the first layer, we first sample across possible hidden states $\boldsymbol{s} \sim \widehat{\boldsymbol{b}}$ 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 $\boldsymbol{a} \sim \widehat{\boldsymbol{\Pi}}_t$ 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 $a_i^* = \max(\boldsymbol{\pi}^i)$ 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 E.

D.3 Dialogue Generation

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

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

We will implement f_{gen} using an LLM, which is fed $\boldsymbol{I}$ 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 I.

E Value Heuristic Implementation Details

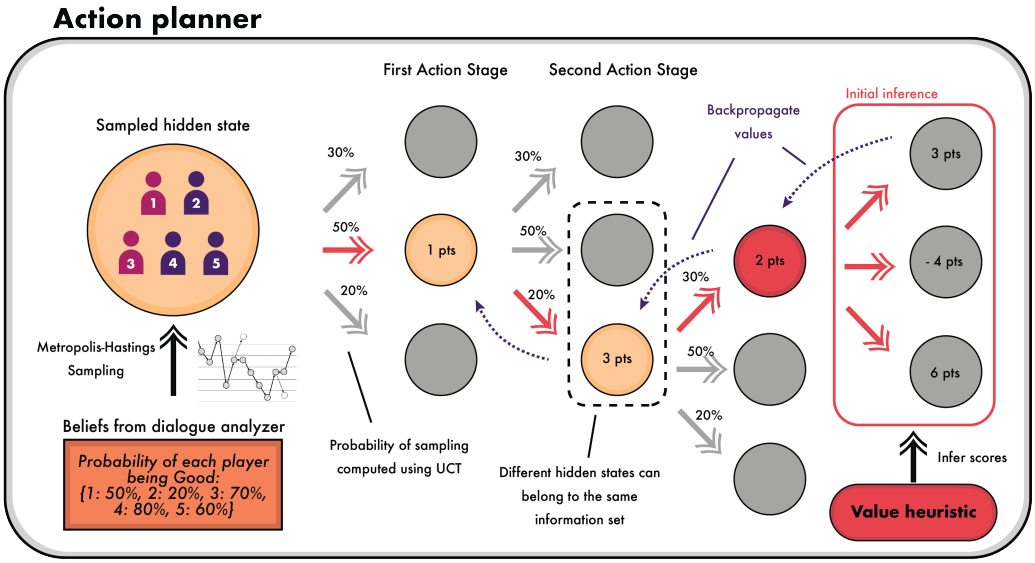


Figure 10: **Overview of how we utilize the trained value heuristics in MCTS tree search to get a non-dialogue based policy**. While we only display the values for a single player in the diagram, note that in practice we infer and update the values for all players at the same time. Next states are sampled 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 10, 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) [10]. 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 backpropogate 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 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, $\widehat{Q}(s, a)$ 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.

$$Q(s, a) = \frac{N(s, a) \cdot Q_{\text{emp}}(s, a) + \alpha \cdot \widehat{Q}(s, a)}{N(s, a) + \alpha} \tag{1}$$

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

F Dialogue Guide Improvement Evaluation Implementation Details

We provide more details on our dialogue improvement evaluation process here and as shown in figure 11. 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 D 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, $\overline{z}_{merlin}$, along with the average z_{evil} scores from the Good players $\overline{z}_{evil}$. We then take the minimum of these two as the feedback score $z = \min\{\overline{z}_{evil}, \overline{z}_{merlin}\}$. 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.

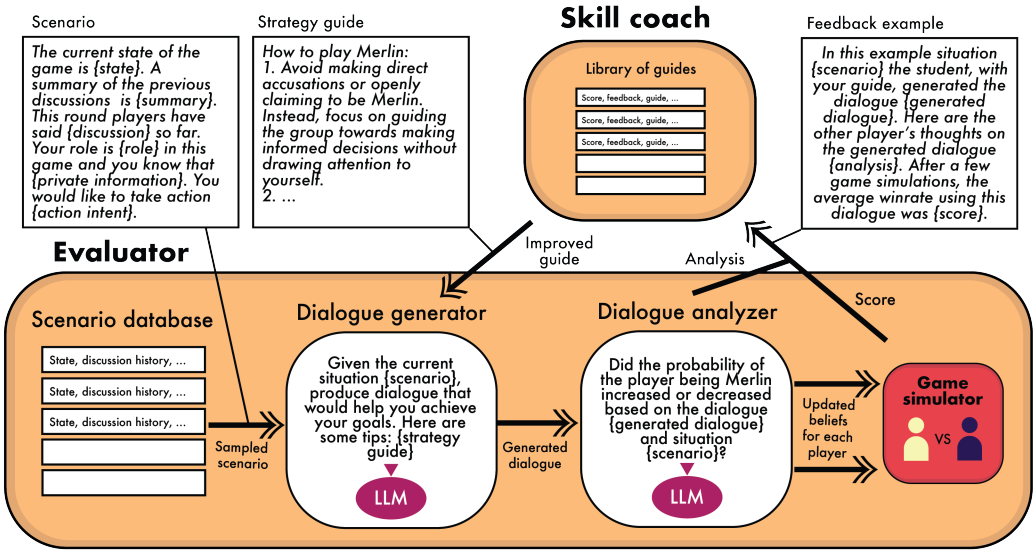


Figure 11: **Overview of our improvement process for learning dialogue generation strategies**. This includes how we evaluate the dialogue and how we collect feedback. The skill coach here can be implemented as either our improvement method, STRATEGIST, or any of the baseline methods we described.

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

G Value Heuristic LLM Prompt and Output Examples

G.1 System Prompts

System prompt 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 runs 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.

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_value1> = value1
    # explanation of what we do next
    ...
    <intermediate_value2> = value2
    # explanation of what we do next
    ...
    player_scores = (player_0_expected_score, player_1_expected_score)
    intermediate_values = {'<intermediate_value1>': intermediate_value1, '<intermediate_value2>':
         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):"

G.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.

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.

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.

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.

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.

G.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>

G.3.1 Example 1

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

GOPS Value Heuristics Function (Before)

```python
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.

GOPS Value Heuristics Function (After)

```python
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}
```

G.3.2 Example 2

Below is the Avalon Value Heuristics Function before improvements.

Avalon Value Heuristics Function (Before)

```python
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.

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
```

G.3.3 Example 3

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

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)

    # 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)
```

```
    # 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

    # Adjust scores based on the difference in the sum of played cards
    adjustment_factor = 0.1  # Adjust this factor based on the level of adjustment desired
    player_0_expected_score += sum_played_cards_diff * adjustment_factor
    player_1_expected_score -= sum_played_cards_diff * adjustment_factor

    # Dynamic adjustment based on uncertainty in the game state
    uncertainty_factor = 0.1 * rounds_left  # Adjust this factor based on the level of uncertainty
          desired
    player_0_expected_score += player_0_potential * uncertainty_factor
    player_1_expected_score += player_1_potential * uncertainty_factor

    # Refine uncertainty factor based on specific conditions
    if len(player_0_hand) == 1:
        uncertainty_factor *= 0.5
    if len(player_1_hand) == 1:
        uncertainty_factor *= 0.5

    # 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,
        'player_0_certain_wins_adjust': player_0_certain_wins_adjust,
        'player_1_certain_wins_adjust': player_1_certain_wins_adjust,
        'sum_played_cards_diff': sum_played_cards_diff,
        'uncertainty_factor': uncertainty_factor
    }

    return (player_0_expected_score, player_1_expected_score), intermediate_values
```

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.

GOPS Value Heuristics Function (After)

```python
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
```

```
    # Adjust scores based on the difference in the sum of played cards
    adjustment_factor = 0.1  # Adjust this factor based on the level of adjustment desired
    player_0_expected_score += sum_played_cards_diff * adjustment_factor
    player_1_expected_score -= sum_played_cards_diff * adjustment_factor

    # Dynamic adjustment based on uncertainty in the game state
    uncertainty_factor = 0.1 * rounds_left  # Adjust this factor based on the level of uncertainty
          desired
    player_0_expected_score += player_0_potential * uncertainty_factor
    player_1_expected_score += player_1_potential * uncertainty_factor

    # Refine uncertainty factor based on specific conditions
    if len(player_0_hand) == 1:
        uncertainty_factor *= 0.5
    if len(player_1_hand) == 1:
        uncertainty_factor *= 0.5

    # 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,
        'player_0_certain_wins_adjust': player_0_certain_wins_adjust,
        'player_1_certain_wins_adjust': player_1_certain_wins_adjust,
        'sum_played_cards_diff': sum_played_cards_diff,
        'uncertainty_factor': uncertainty_factor
    }

    return (player_0_expected_score, player_1_expected_score), intermediate_values
```

G.3.4 Example 4

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

GOPS Value Heuristics Function (Before)

```python
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.

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)

```python
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)

# Introduce a penalty factor for excessively high potential scores
penalty_threshold = 100  # Define a threshold for potential scores to trigger penalty
penalty_factor = 0.5  # Define a factor by which to reduce potential scores above threshold

if player_0_potential > penalty_threshold:
   player_0_potential = penalty_threshold + (player_0_potential - penalty_threshold) *
     penalty_factor

if player_1_potential > penalty_threshold:
   player_1_potential = penalty_threshold + (player_1_potential - penalty_threshold) *
     penalty_factor

# 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
```

H 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.

H.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.

H.2 Idea Generation Examples

The following is the selected dialouge 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.

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."

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.

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

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.

H.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.

H.3.1 Example 1

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. 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 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. 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.

H.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.

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.

H.3.3 Example 3

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.
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 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.
9. Analyze the voting patterns and craft a statement subtly highlighting any inconsistencies or suspicious trends in the voting behavior of specific players.

I 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.

I.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.

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]

I.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.

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 questionaire filled out above.

Prompt for speech generation

```
<System prompt>
<Filled out worksheet>
```

Assemble your responses to the questionaire 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.

I.3 Dialogue Analysis

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

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

Prompt for dialogue analysis: Merlin

```
<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 Merlin (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 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 4's speech seems generally clueless and lacks any strategic insight, which decreases the probability of Player 4 being Merlin.

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

Generated dialogue analysis follows. The role is Merlin.

LLM generated dialogue analysis: Merlin

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

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

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

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')}

J Other Ablation Studies

J.0.1 Base LLM Model

We show the performance of different base models in Figure 12.

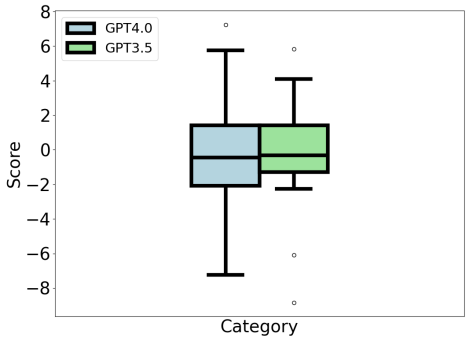


Figure 12: Game play performance of base LLM models on six card GOPS: GPT3.5 and GPT4.0. We see that the two language models perform similarly with our method.

J.0.2 Search Budget

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

Table 4: Average score difference for MCTS (num_rollout=32) + LLMFunction (Player1, top-3 functions shown in the table) vs. MCTS (num_rollout=32) + RandomRollout (Player2, num_rollout=10); 100 games for each experiment;

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

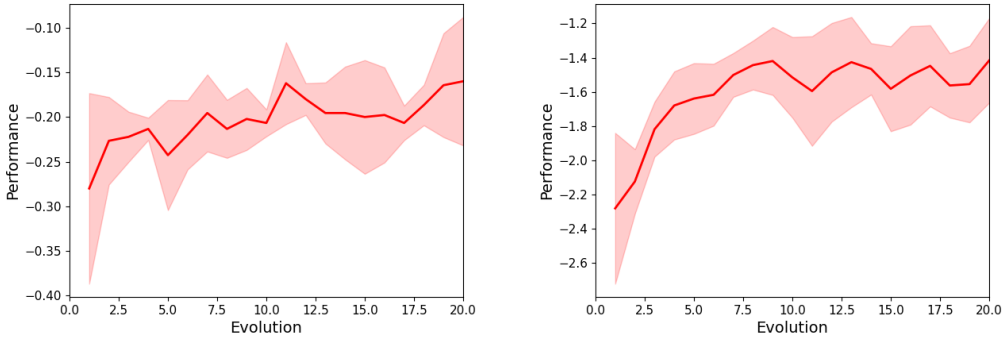
K Details on Learning the Value Heuristic via Reinforcement Learning

We employ Monte-Carlo based RL approach ([34]) 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.,

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

where $\mathcal{N}$ represents the number of actors, $V_\theta^i(s_t), i = 1, 2, \cdots, \mathcal{N}$ denotes the value function for each actor, and T is the time horizon. Notice that s_t and $Score^i(s_t)$ 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 $Score^i(s_t)$ (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



(a) Avalon (b) GOPS

Figure 13: 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.

Table 5: 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

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 5.

Having introduced the set up, one can observe in Figure 13 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.

L 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.