# SPOON: STRATEGY-BASED PROMPT OPTIMIZATION FOR MULTI-AGENT LLM GAMES

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

LLM agents can collaborate, negotiate, and compete in multi-agent settings where each player's success depends on the actions of others. While multi-agent games provide a useful formalization of these settings, improving LLM agents' performance in these games often relies on reinforcement learning, requiring access to model parameters and costly training across many optimization iterations. To address this, we propose **SPOON**, a strategy-based prompt optimizer that represents each player's strategy as explicit rules written in text and refines them without updating model weights. Our strategic abstractions compress many game histories into shared states, allowing one rule update to improve behavior across multiple situations. Explicit strategies enable our method to examine how both players' rules interact and guide structured best-response search, directly improving each player's strategy in response to another as they co-evolve across optimization iterations. Across 19 existing benchmarks spanning classical multi-agent games, practical multi-agent applications, and an extension to prompt optimization for single-agent tasks, our method improves optimization efficiency and held-out performance compared with training-based and prompt optimization baselines.

## 1 INTRODUCTION

LLMs are increasingly used in multi-agent settings that require collaboration, negotiation, and competition (Li et al., 2023; Wu et al., 2023; Guertler et al., 2025a; Yao et al., 2025). These settings can be modeled as games because players pursue objectives whose outcomes depend on both their own strategies and those of other players (Osborne & Rubinstein, 1994). Learning effective strategies in these games therefore requires accounting for other players' strategies, a central focus of multi-agent learning (Shoham et al., 2007). To learn such strategies, training-based approaches use self-play and multi-agent reinforcement learning (MARL), updating policy parameters through repeated play among co-evolving players (Silver et al., 2017; Lanctot et al., 2017; Liu et al., 2025a).

Prompt optimization offers an alternative to RL for improving LLM performance by revising natural-language prompts while keeping model weights fixed. It requires no access to model parameters or gradients, making it applicable to closed models. Furthermore, natural-language reflection supports sample-efficient optimization by identifying why a rollout failed and translating that diagnosis directly into a prompt update. This extracts actionable guidance from the trajectory beyond its scalar reward (Pryzant et al., 2023; Yuksekgonul et al., 2024; Agrawal et al., 2025). The resulting prompts also offer interpretability and portability: they can be inspected, edited by humans, and applied to other models without transferring model weights.

We propose **SPOON**, a prompt optimizer that improves LLM agents' performance in multi-agent games by revising explicit textual strategies. Osborne & Rubinstein (1994) describe a strategy as a plan specifying how a player acts given the information available to it. We represent each strategy through natural-language conditions that describe when a rule applies and behavioral rules that specify what to do. For example, in the card game Briscola, a player may hold different cards in two tricks yet face the same problem: none can beat the opponent's card. Although the hands and preceding actions differ, both situations satisfy the condition "the current trick cannot be won." We call this shared situation a strategic state and pair it with a rule, such as "discard the lowest-point card." Revising this rule can improve decisions across different hands without requiring a separate correction for each card combination. This abstraction allows a small set of rules to guide play



Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

across many game histories, including those not encountered during optimization. This design draws on evidence from negotiation training that learning the common structure of different situations helps people reuse decision principles in unfamiliar interactions (Nadler et al., 2003; Moran et al., 2008; Gentner et al., 2003).

Our explicit strategy representation also supports structured best-response search. In game theory, a best response maximizes a player's payoff given the other player's strategy (Osborne & Rubinstein, 1994). Because both strategies are explicitly written in text, we propose a method that directly examines how their decision rules interact and uses this reasoning to guide targeted changes to one player's strategy in response to another.

We evaluate explicit strategy optimization across classical multi-agent games, practical multi-agent applications, and an extension to prompt optimization for single-agent tasks. We first study five games from the MEMO suite, covering negotiation and strategic decision making under perfect and imperfect information (Xie et al., 2026). We then study practical applications spanning adversarial jailbreaking and reward judging, alongside cooperative meta-reasoning where a planning agent guides a reasoning agent on mathematical problems. Motivated by progress in prompt optimization (Agrawal et al., 2025), we extend our game formulation to single-agent instruction following, question answering, and evidence retrieval by introducing adaptive adversaries during optimization. Across these settings, our method improves optimization efficiency and held-out performance over the relevant training-based and prompt optimization baselines.

In summary, we make three contributions. (i) We introduce explicit strategic abstractions as the unit of optimization, enabling efficient learning and generalization through shared decision rules. (ii) We develop a method that uses both players' explicit strategies to guide structured best-response search, proposing changes to one player's strategy in response to another. (iii) We demonstrate improved optimization efficiency and held-out performance over training-based and prompt optimization baselines across classical games, practical multi-agent applications, and single-agent tasks.

## 2 LLM Agents as Game Players

### 2.1 Game Formulation

We formulate sequential interactions between players as extensive-form games (Osborne & Rubinstein, 1994). We consider a two-player game, using $i$ to denote one player and $-i$ to denote the other. At decision step $t$, player $i$ chooses an action $a_t$ using its observed history $h_t^i$. This history contains the information available to the player, including private information and previously observed actions. For each observed history $h_t^i$, player $i$'s *behavioral strategy* specifies a distribution over legal actions, $\pi_i(a_t \mid h_t^i)$.

Together with chance events, the two players' strategies determine a distribution over complete game histories, or trajectories $\tau$. Given a payoff $u_i(\tau)$ for each trajectory, player $i$ seeks to maximize its expected payoff

$$J_i(\pi_i, \pi_{-i}) = \mathbb{E}\big[u_i(\tau)\big], \tag{1}$$

where the expectation is over chance events and actions sampled under both strategies. The payoff functions can describe competing, shared, or partially aligned objectives. A *best response* for player $i$ is a strategy $\pi_i$ that maximizes player $i$'s expected payoff against the other player's fixed strategy:

$$\mathrm{BR}_i(\pi_{-i}) = \arg\max_{\pi_i} J_i(\pi_i, \pi_{-i}). \tag{2}$$

In our LLM game setting, each player's model conditions on its observed history $h_t^i$ and a textual strategy supplied in its context window to generate actions according to $\pi_i(a_t \mid h_t^i)$. Each generated response specifies an action $a_t$, and the task's evaluation rule assigns a payoff $u_i(\tau)$ to the completed interaction. We optimize each player's strategy by revising its textual description in the model's prompt, while keeping model weights fixed.

Our LLM players co-evolve through repeated play, with each update seeking to improve one player's textual strategy against another's current strategy. Once a player responds effectively to a fixed strategy, further play against that same strategy largely repeats challenges it already handles, providing limited feedback about its remaining weaknesses. Updating both players together changes the challenges they present to one another, creating an adaptive curriculum as their strategies evolve. We

use this adaptive curriculum to improve optimization efficiency by concentrating interactions on unresolved weaknesses, and to improve held-out generalization through exposure to a broader range of strategies.

### 2.2 Explicit Strategies for Efficient Optimization and Generalization

A game-theoretic perspective can help improve optimization efficiency and generalization in multi-agent LLM games. We next explain how our explicit textual strategy representation supports these gains through strategic abstraction and structured best-response search.

**Strategic abstraction makes improvements localized and generalizable**. The number of possible game histories grows rapidly with interaction length, although many histories share the same underlying strategic problem. We seek to group histories that are *strategically equivalent* for selecting a behavioral rule: a common rule can address their shared strategic problem despite differences in surface details. For player $i$, we define an abstraction from its possible observed histories $\mathcal{H}_i$ to a set of strategic states:

$$\phi_i : \mathcal{H}_i \to \{S_1, \ldots, S_{K_i}\}, \qquad K_i \ll |\mathcal{H}_i|. \tag{3}$$

The aim is to represent many possible histories using a small number of strategic states. Each state is described in natural language and paired with a behavioral rule. The grouping aims to preserve distinctions that matter for how actions affect payoffs, following the motivation of state abstraction in sequential decision making (Li et al., 2006). An improvement to a shared rule can therefore generalize to other histories where that rule applies. The representation provides four benefits. The first three improve optimization efficiency; the fourth supports generalization at test time:

1. *Localizing failures.* A strategy contains multiple conditions and behavioral rules, so a failure can arise from one part even when other parts remain effective. Explicit representation lets us attribute the failure to the particular condition or rule that needs revision.
2. *Targeting the search.* Once a weakness is localized, search can focus on revising the affected condition or rule rather than exploring arbitrary changes to the entire strategy. This reduces the space of candidate edits.
3. *Verifying local improvements.* Because we change only a local part of the strategy, we only need to verify interactions involving that part, keeping verification targeted and inexpensive.
4. *Generalizing at test time.* A behavioral rule captures a strategic pattern shared by multiple histories. At test time, the revised rule can guide responses to unseen histories with the same structure, extending improvements beyond the interaction that motivated the update.

This form of localized optimization relies on explicit, separately editable conditions and behavioral rules. In contrast, parameter-based RL learns a player's strategy implicitly through updates to the LLM's weights. Without an explicit representation of these strategic components, RL cannot directly isolate and revise a particular rule.

**Explicit strategies support structured best-response search**. Our textual representation makes both players' strategies directly available for inspection. In contrast, when RL encodes strategies only in model weights, their decision rules remain implicit, and the other player's strategy is typically inferred from observed play. With explicit access to both strategies, our method performs structured best-response search directly against the other player's strategy. Comparing both strategies helps identify weaknesses in one strategy relative to the other, allowing the search to focus on addressing those weaknesses rather than exploring arbitrary strategy revisions.

## 3 Strategy-Based Prompt Optimization

Our method combines an explicit strategy representation with an optimization loop that improves players' strategies through repeated play. Each player's strategy groups game histories into strategic states paired with behavioral rules, and the optimizer uses both players' explicit strategies to guide structured best-response search.

### 3.1 STRATEGY REPRESENTATION

For player $i$, we represent its textual strategy $\Pi_i$ as $K_i$ strategic states paired with behavioral rules:

$$\Pi_i = \{(S_k, A_k)\}_{k=1}^{K_i}. \tag{4}$$

Each state $S_k$ is defined by a natural-language condition on the player's available history $h_t^i$ at decision step $t$. The condition groups histories that differ in their details but share a strategic problem that the same behavioral rule can address. The associated behavioral rule $A_k$ gives natural-language instructions for choosing an action when this condition holds.

At each decision point, player $i$ selects one strategic state whose condition applies to its available history. We call this selection *routing* and describe it by a function $r_i(h_t^i; \Pi_i)$. The function returns a number $k \in \{1, \ldots, K_i\}$ identifying the selected state $S_k$ and its associated rule $A_k$. The player's frozen model $M_i$ then uses this rule together with $h_t^i$ to generate an action appropriate to the current situation. These two steps determine the player's action distribution $\pi_i$:

$$k = r_i(h_t^i; \Pi_i), \qquad \pi_i(\cdot \mid h_t^i; \Pi_i) = M_i(\cdot \mid h_t^i, A_k). \tag{5}$$

### 3.2 OPTIMIZATION LOOP

At each iteration, we play a game, select a player to update, localize its failure, and propose changes to its strategy. We accept at most one update that passes replay checks for payoff improvement on the current game and no payoff loss on affected stored examples. Algorithm 1 summarizes the loop.

**Play and player selection**. At iteration $n$, we sample an instance $x_n$ from the optimization set $\mathcal{D}_{\text{opt}}$ and play using the current strategies $(\Pi_1, \Pi_2)$, obtaining a trajectory $\tau_n$ and payoffs $u_1(\tau_n), u_2(\tau_n)$. We select the losing player to update in competitive games; ties and cooperative games use a task-specific selection rule. Let $i$ denote the selected player.

**Localizing failures and proposing deviations**. An *optimizer LLM* receives the current game's trajectory $\tau_n$ and both players' strategies $(\Pi_i, \Pi_{-i})$. Access to both strategies lets it reason about how one player's strategy should change in response to the other. It proposes which condition or behavioral rule to revise based on its diagnosis of the failure. While the optimizer uses both strategies to propose updates, neither player is given the other player's strategy during play. During both optimization and evaluation, player $i$ receives only its own textual strategy $\Pi_i$ and observed history $h_t^i$.

In game theory, a *unilateral deviation* changes one player's strategy while holding the other player's strategy fixed. We search for such deviations through localized revisions to $\Pi_i$, keeping $\Pi_{-i}$ fixed throughout the update attempt. We call each proposed revision a *strategic deviation* $\delta$ and write $\Pi_i'$ for the resulting candidate strategy. The optimizer proposes one of three forms:

1. **Response edit.** The selected state $S_k$ is appropriate, but its rule gives a poor response. Revise $A_k$ while keeping the state condition unchanged.
2. **Rerouting.** The history selects an inappropriate state $S_j$, while another existing state $S_k$ matches the situation. Revise their conditions so that the history selects $S_k$ and follows $A_k$. If $A_k$ also needs improvement, revise it as part of the proposal.
3. **New strategic state.** No existing state captures the distinction needed for an appropriate response. Add a condition and rule $(S_{\text{new}}, A_{\text{new}})$, adjusting existing conditions to keep state selection unambiguous.

The optimizer proposes up to $B$ candidate deviations per iteration. Failed verification can guide subsequent proposals.

**Verifying candidate deviations**. We test each candidate deviation by replaying the current game with the updated strategy $\Pi_i'$ against the other player's current strategy $\Pi_{-i}$. We check that the updated strategy selects the intended strategic state and improves player $i$'s payoff over $\Pi_i$ under the same replay conditions.

To prevent new updates from undoing earlier improvements, each player maintains a *strategic frontier* $F_i$ of saved examples. For each strategic state $S_k$ improved by a previously accepted update, we

retain one replayable game example $e_k$ showing that improvement, called a *witness*. Each witness includes a saved copy of the other player's strategy from that game.

We check only witnesses associated with strategic states that the candidate deviation may affect. For each witness, we compare the updated and current strategies under the same game conditions, against the saved copy of the other player's strategy. The updated strategy must preserve the witness's intended state assignment and achieve at least the same payoff. This provides a local regression check focused on the affected parts of the strategy.

A candidate deviation passes verification if it selects the intended strategic state and improves player $i$'s payoff on the current game, while preserving payoff on the checked frontier witnesses.

**Accepting an update**. If any candidate deviations pass verification, we accept the one with the largest payoff improvement on the current game. When deviations give the same payoff improvement, we prefer changing existing states over adding a new state. This avoids expanding the strategy when its existing states can support an equally effective update. The verified game becomes the witness for the state improved by the accepted update, replacing its previous witness if one exists. If no candidate deviation passes verification, both the strategy and frontier remain unchanged.

**Algorithm 1** Game-based optimization of explicit strategies

**Require:** Initial strategies $(\Pi_1, \Pi_2)$, optimization set $\mathcal{D}_{\text{opt}}$, iterations $N$, proposal budget $B$, player-selection rule

$F_1, F_2 \leftarrow \emptyset$
**for** $n = 1, \ldots, N$ **do**
  Sample $x_n$ from $\mathcal{D}_{\text{opt}}$ and play using $(\Pi_1, \Pi_2)$
  Obtain $\tau_n$ and payoffs; select player $i$ to update
  Optimizer LLM proposes where to edit using $(\tau_n, \Pi_i, \Pi_{-i})$
  Initialize the set of verified deviations $\mathcal{V} \leftarrow \emptyset$
  **for** $b = 1, \ldots, B$ **do**
    Propose deviation $\delta_b$ and candidate strategy $\Pi_i'$ using the diagnosis and verification feedback
    Replay the current game with $\Pi_i'$ and $\Pi_i$ against $\Pi_{-i}$ under matched conditions
    Select frontier witnesses associated with affected strategic states
    Replay those witnesses with $\Pi_i'$ and $\Pi_i$ against their saved other-player strategies under matched conditions
    **if** routing and payoff-improvement checks pass on the current game, and witness payoffs do not decrease **then**
      Save the deviation, payoff improvement, and verified game in $\mathcal{V}$
    **end if**
  **end for**
  **if** $\mathcal{V} \neq \emptyset$ **then**
    Select the deviation with the largest payoff improvement
    For equal improvements, prefer changing existing states over adding a new state
    Update $\Pi_i$ with the selected deviation
    Replace or initialize the improved state's witness using the verified game and $\Pi_{-i}$
  **else**
    Leave $\Pi_i$ and $F_i$ unchanged
  **end if**
**end for**
**return** $(\Pi_1, \Pi_2)$

## 4 EXPERIMENTS

We evaluate SPOON across classical game environments, practical multi-agent applications, and an extension to prompt optimization for single-agent tasks. We first use classical games as controlled settings for evaluating strategic decision making under different rules and information constraints. We then test SPOON's applicability beyond classical games in practical use cases, including adversarial jailbreaking, reward judging, and cooperative mathematical reasoning. Finally, we examine

Table 1: Win rates (%) with Qwen2.5-7B-Instruct. Baselines reported from Xie et al. (2026).

| | Negotiation | | Imperfect info. | | Perfect info. | |
|---|---|---|---|---|---|---|
| Method | Simple Negotiation | TwoDollar | Kuhn Poker | Briscola | SimpleTak | Mean |
| *Static prompting* | | | | | | |
| Default | 24.00 | 17.10 | 45.30 | 2.80 | 15.10 | 20.90 |
| CoT | 23.80 | 18.70 | 42.00 | 25.80 | 13.60 | 24.80 |
| ToT | 27.10 | 20.70 | 42.20 | 22.70 | 15.10 | 25.60 |
| *Prompt and context optimization* | | | | | | |
| TextGrad | 37.10 | 29.30 | 52.80 | 7.10 | 22.40 | 29.90 |
| MIPRO | 42.40 | 47.50 | 53.80 | 2.20 | 20.90 | 33.40 |
| GEPA | 34.40 | 31.70 | 55.80 | 3.30 | 19.30 | 28.80 |
| MEMO | 48.00 | 48.40 | 60.00 | 31.10 | 34.00 | 44.30 |
| *Self-play reinforcement learning* | | | | | | |
| UnstableBaseline | 41.10 | 30.40 | 52.70 | 53.30 | **47.30** | 45.00 |
| SPIRAL | 45.70 | – | 56.70 | – | 32.70 | – |
| **SPOON (ours)** | **49.33** | **50.67** | **64.00** | **69.33** | 42.00 | **55.07** |

whether introducing adaptive adversaries extends SPOON to single-agent language tasks, including instruction following, question answering, and evidence retrieval. Across these settings, we assess optimization efficiency by the number of optimization episodes used to learn the strategies, and generalization through performance on held-out opponents or task instances. We compare with relevant training-based and prompt optimization baselines and use ablations to examine the contributions of SPOON's components.

### 4.1 Classical Game Environments

**Tasks and Baselines**. We study five negotiation, card, and board game environments evaluated by MEMO (Xie et al., 2026): SimpleNegotiation, TwoDollar, KuhnPoker, Briscola, and SimpleTak.

MEMO is a prompt optimizer for LLM games that uses lessons from past play to evolve candidate prompts. SPOON represents each player's strategy as explicit strategic states and behavioral rules, allowing the optimizer to examine how both players' strategies interact and identify specific conditions or rules to revise.

We report the published Qwen2.5-7B-Instruct results from MEMO for three groups of baselines. Static prompting includes default prompting, chain-of-thought (CoT), and tree-of-thought (ToT). Prompt and context optimization includes TextGrad (Yuksekgonul et al., 2024), MIPRO (Opsahl-Ong et al., 2024), GEPA (Agrawal et al., 2025), and MEMO. UnstableBaseline and SPIRAL update model weights through self-play reinforcement learning.

**Experimental Setup**. We use MEMO's game environments, opponent prompts, and scoring. Both players and the optimizer use frozen Qwen2.5-7B-Instruct with greedy decoding. We run 30 optimization episodes in each of the five game environments. We evaluate the learned strategies against three held-out opponents: Qwen3-235B-A22B-Instruct-2507, Gemini-2.5-Flash-Lite, and Grok 4.3. We replace MEMO's Grok-4-Fast-Non-Reasoning with Grok 4.3 because the former was unavailable through OpenRouter at evaluation time. We repeat the procedure with three random seeds and average win rates across seeds and opponents.

**Results**. Table 1 shows that SPOON achieves a mean held-out win rate of 55.07%, higher than the published means for the prompt optimization baseline MEMO (44.30%) and the RL baseline UnstableBaseline (45.00%). This performance is achieved with 30 optimization games in each of the 5 environments. For comparison, MEMO reports 2,000 self-play games per environment, while UnstableBaseline uses 38,000 games for RL training on KuhnPoker (Xie et al., 2026).

Table 2: Defender safety and benign non-refusal rates (%) with Qwen2.5-7B-Instruct. Baselines from Liu et al. (2025b). "Comply" denotes non-refusal; WJB denotes WildJailbreak.

| Method | HarmBench ASR ↓ | WJB Comply ↑ | XSTest Comply ↑ |
|---|---|---|---|
| *Published references* | | | |
| Base Qwen | 27.80 | **99.20** | 94.80 |
| Self-RedTeam | **16.10** | 97.90 | **96.00** |
| **SPOON** | 17.20 | 96.00 | 95.60 |

Table 3: RewardBench preference accuracy (%, non-voting) with Qwen3-8B. Baselines from Xu et al. (2026).

| Method | Chat ↑ | Chat-Hard ↑ |
|---|---|---|
| *Published references* | | |
| Rubric prompting | 73.90 | 63.60 |
| Rubric-RM (SFT) | 88.20 | 74.10 |
| Rubric-ARM (RL) | **89.40** | 79.60 |
| **SPOON** | 88.27 | **80.26** |

### 4.2 Practical Applications of Multi-Agent Interaction

We next test SPOON in practical applications beyond classical games: adversarial jailbreaking, reward judging, and cooperative mathematical reasoning. We use existing RL methods as reference settings, following their player roles, interaction sequences, and model backbones. The goal is to isolate the effect of replacing RL weight updates with explicit strategy optimization on held-out performance and data efficiency.

#### 4.2.1 Adversarial Jailbreaking

Adversarial jailbreaking seeks to elicit unsafe responses from an LLM; effective defenses must resist these attacks while continuing to answer benign requests. We follow Self-RedTeam's attacker–defender interaction (Liu et al., 2025b), using Qwen2.5-7B-Instruct for both players. The attacker rewrites harmful requests to elicit unsafe assistance and rewrites benign requests to induce unnecessary refusals. The defender seeks safe responses and appropriate refusal, with outcomes assessed by fixed WildGuard-7B judgments. Following Self-RedTeam, optimization balances harmful and benign WildJailbreak requests, each presented directly or rewritten by the attacker. While Self-RedTeam updates model weights through RL, SPOON optimizes separate explicit strategies for the two roles with model weights frozen.

We evaluate the frozen defender on 1,500 fixed HarmBench attacks and 250 benign prompts each from WildJailbreak and XSTest. HarmBench measures attack success rate, where lower is better; the benign benchmarks measure non-refusal, where higher is better. Table 2 compares SPOON with the published base model and Self-RedTeam. Self-RedTeam uses 26,000 WildJailbreak seed requests for RL, together with a 30,000-example auxiliary SFT dataset. SPOON achieves broadly comparable safety and benign compliance using only 1,000 seed requests selected from the same WildJailbreak training set.

#### 4.2.2 Rubric-Based Preference Judging

Rubric-based preference judging uses request-specific criteria to determine which of two candidate answers is better. We follow Rubric-ARM's cooperative two-player structure (Xu et al., 2026): a rubric generator observes the request and writes evaluation criteria, and a judge uses these criteria to compare two fixed answers. Both players use Qwen3-8B backbones initialized from released Rubric-RM SFT checkpoints (Liu et al., 2025c). We retain the generator–judge interaction and released correctness and judge-format rewards. While Rubric-ARM alternates RL updates to the judge and generator, SPOON alternates updates to their explicit strategies with model weights frozen.

We evaluate non-voting preference accuracy on RewardBench (Lambert et al., 2025) Chat (358 pairs) and Chat-Hard (456 pairs). Table 3 includes the published rubric-prompting and Rubric-RM SFT baselines, with Rubric-ARM as the primary RL comparison. SPOON obtains comparable Chat accuracy and slightly higher Chat-Hard accuracy using only 1,792 examples selected from the same general-domain OpenRubrics training data used by Rubric-ARM. The Rubric-ARM paper does not report the size of its RL training set.

Table 4: Mathematical reasoning accuracy (%) with Qwen2.5-7B-Instruct. Baselines from Wan et al. (2025). Mean is the unweighted average across the seven benchmarks.

| Method | MATH 500 | GSM8K | AIME24 | AMC23 | GaoKao 2023En | Minerva Math | Olympiad Bench | Mean |
|---|---|---|---|---|---|---|---|---|
| *Published references* | | | | | | | | |
| CoT | 75.00 | **92.04** | 6.67 | 47.50 | 56.62 | **35.66** | 38.22 | 50.24 |
| CoT + RL | **77.20** | 91.36 | 6.67 | 50.00 | 54.81 | 34.93 | 38.37 | 50.48 |
| Plan-and-solve + RL | 76.40 | 91.81 | 10.00 | 52.50 | 55.06 | 32.35 | 37.78 | 50.84 |
| ReMA + RL | 74.40 | 90.60 | **20.00** | 57.50 | 57.92 | 34.93 | 36.30 | 53.09 |
| **SPOON** | 76.00 | 91.96 | 16.67 | **60.00** | **62.86** | 34.93 | **39.85** | **54.61** |

Table 5: Single-agent test scores (%) with Qwen3-8B. Baselines: Agrawal et al. (2025); Feedback Descent: Lee et al. (2025).

| Method | IFBench | HotpotQA | HoVer |
|---|---|---|---|
| Unoptimized | 36.90 | 42.33 | 35.33 |
| GRPO | 35.88 | 43.33 | 38.67 |
| MIPROv2 | 36.22 | 55.33 | 47.33 |
| GEPA | 38.61 | 62.33 | 52.33 |
| Feedback Descent | 38.78 | 60.00 | **60.00** |
| **SPOON (ours)** | **42.86** | **64.67** | 53.33 |

#### 4.2.3 Cooperative Mathematical Reasoning

ReMA (Wan et al., 2025) uses multi-agent RL to train a meta-thinking agent that guides a reasoning agent on mathematical problems. The planner observes a problem and produces a high-level solution plan; the solver receives the problem and plan, then generates a solution checked by a fixed mathematical grader. We follow this single-turn planner–solver structure, using Qwen2.5-7B-Instruct for both players and the strategy optimizer. While ReMA trains the two policies with RL, SPOON alternates updates to their explicit strategies while keeping model weights fixed.

We optimize on a subset of the training split of MATH (Hendrycks et al., 2021), the competition mathematics benchmark used to train ReMA. After optimization, we evaluate one fixed strategy pair across the seven held-out benchmarks in Table 4. Our primary comparison is ReMA + RL, which trains separate planner and solver policies. We also include its published single-model baselines: CoT uses chain-of-thought prompting without additional training; CoT + RL trains this reasoning policy with RL; and Plan-and-solve + RL trains one model to generate a plan followed by a solution within a single response. SPOON achieves the highest mean accuracy among the compared methods and outperforms ReMA on most benchmarks. These results use only 512 problems selected from the same 7,500-problem MATH training set used by ReMA.

### 4.3 Extension to Single-Agent Tasks

We examine whether introducing an adaptive adversary through SPOON improves performance on single-agent language tasks, including instruction following, question answering, and evidence retrieval.

**Tasks and Baselines**. We follow GEPA's task setups for instruction following on IFBench, question answering on HotpotQA, and evidence retrieval on HoVer. We report the Qwen3-8B results for the unoptimized pipeline, GRPO, MIPROv2, and GEPA from Agrawal et al. (2025), and the Feedback Descent results from Lee et al. (2025). GRPO updates model weights, while the prompt optimizers keep them fixed.

**Experimental Setup**. We retain GEPA's Qwen3-8B backbone, task modules, retrieval tools, and decoding settings. SPOON adds an explicit IF/THEN strategy to the solver, with each module following a rule selected according to its current input. During optimization, an adaptive adversary modifies training requests to seek harder instances that expose weaknesses in the solver's strategy. Modified requests first undergo validity checks. For IFBench, the adversary proposes instruction requirements, such as a word-count limit. Our code includes these requirements in the modified

Table 6: Ablation win rates (%), averaged across three seeds and three held-out opponents, with 50 games per seed–opponent pair.

| Variant | Simple Negotiation | TwoDollar | Kuhn Poker | Briscola | SimpleTak | Mean |
|---|---|---|---|---|---|---|
| **SPOON** | **49.33** | **50.67** | **64.00** | **69.33** | **42.00** | **55.07** |
| Unstructured strategy | 43.11 | 45.56 | 58.44 | 46.89 | 36.22 | 46.04 |
| Own-strategy access only | 45.33 | 47.56 | 60.00 | 56.00 | 38.67 | 49.51 |
| Fixed opponent | 42.00 | 44.67 | 56.89 | 51.33 | 34.89 | 45.96 |

request and sets GEPA's instruction checkers to test those same requirements. We accept the request only if an example answer supplied by the adversary passes every check, demonstrating that the requirements can be satisfied together. The solver receives only the request, without the example answer, and its response is scored using the same checks. For HotpotQA and HoVer, rewrites must pass semantic checks for consistency with the original meaning and reference answer or evidence. GEPA's task checkers then score the solver using the unchanged reference answers or supporting documents. The solver is rewarded for task success, while the adversary receives the negative of the solver's reward, making the interaction a zero-sum game. SPOON optimizes both players' textual strategies with model weights frozen.

For each task, we pool GEPA's 150 training and 300 validation examples into a single optimization set of 450 examples. After optimization, we freeze the solver's strategy and remove the adversary. We evaluate on the same held-out test sets used by GEPA, using its scoring code: 294 examples for IFBench and 300 each for HotpotQA and HoVer.

**Results**. SPOON outperforms GEPA on all three tasks and achieves the highest IFBench and HotpotQA scores among the methods in Table 5. These results show that SPOON's adversarial strategy optimization can improve performance on held-out single-agent tasks.

### 4.4 Ablations and Analysis

We ablate three components on all five MEMO games using the setup in Section 4.1. Table 6 reports win rates averaged over three seeds and the same three opponents.

**Strategic Abstraction**. We replace the explicit state–rule strategy representation $\Pi_i = \{(S_k, A_k)\}$ with a generic single prompt describing how to play the game. The model updates this prompt without a required structure. The optimizer still sees both strategies and the trajectory; co-evolution and payoff checks are retained. Lower win rates across all five games support the explicit state–rule representation.

**Access to Both Strategies**. When updating player $i$, the optimizer receives $(\tau_n, \Pi_i)$ instead of $(\tau_n, \Pi_i, \Pi_{-i})$. Strategy representation, co-evolution, verification, and the executing players' information remain unchanged. The consistent performance decrease indicates that inspecting the opponent's strategy provides useful guidance for optimization beyond observing its actions in the trajectory.

**Co-evolution**. Following MEMO's fixed-opponent setup, we update only one player's strategy while keeping the opponent fixed. The opponent uses Qwen2.5-7B-Instruct with MEMO's default game prompt. The optimizer still sees both players' textual strategies. Lower win rates against held-out opponents support the benefit of co-evolution for generalization.

## 5 Conclusion

We introduced SPOON, a strategy-based prompt optimizer for multi-agent LLM games that keeps model weights fixed. SPOON groups histories into strategic states with shared rules and uses both players' explicit strategies to guide structured best-response search as they co-evolve. Experiments show higher mean win rates on classical games and comparable or better performance on practical multi-agent applications using small optimization sets. With adaptive adversaries, SPOON also outperforms GEPA on three single-agent tasks.

### AI USE STATEMENT

We used generative AI tools to write code for implementing and evaluating our method, identify relevant literature, and assist with writing and revising the manuscript. The authors reviewed the generated code for consistency with the intended implementation and reviewed and revised the AI-assisted text. We take responsibility for the final content of this work, including its text, claims, code, and results.

### REPRODUCIBILITY STATEMENT

Section 3 describes SPOON's strategy representation and optimization procedure, summarized in Algorithm 1. Section 4 describes the model backbones, benchmark setups, optimization data, evaluation procedures, and sources of published baseline results.

## REFERENCES

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning. *arXiv preprint arXiv:2507.19457*, 2025. URL https://arxiv.org/abs/2507.19457.

Dedre Gentner, Jeffrey Loewenstein, and Leigh Thompson. Learning and transfer: A general role for analogical encoding. *Journal of Educational Psychology*, 95(2):393–408, 2003. doi: 10.1037/0022-0663.95.2.393. URL https://groups.psych.northwestern.edu/gentner/papers/GentnerLoewensteinThompson03.pdf.

Leon Guertler, Bobby Cheng, Simon Yu, Bo Liu, Leshem Choshen, and Cheston Tan. TextArena. *arXiv preprint arXiv:2504.11442*, 2025a. URL https://arxiv.org/abs/2504.11442.

Leon Guertler, Tim Grams, Zichen Liu, and Bobby Cheng. UnstableBaselines. Zenodo, 2025b. URL https://doi.org/10.5281/zenodo.15719271. Software, version 0.1.0.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. *arXiv preprint arXiv:2103.03874*, 2021. URL https://arxiv.org/abs/2103.03874.

Jakub Grudzien Kuba, Mengting Gu, Qi Ma, Yuandong Tian, Vijai Mohan, and Jason Chen. Language self-play for data-free training. *arXiv preprint arXiv:2509.07414*, 2025. URL https://arxiv.org/abs/2509.07414.

Nathan Lambert, Valentina Pyatkin, Jacob Morrison, LJ Miranda, Bill Yuchen Lin, Khyathi Chandu, Nouha Dziri, Sachin Kumar, Tom Zick, Yejin Choi, Noah A. Smith, and Hannaneh Hajishirzi. RewardBench: Evaluating reward models for language modeling. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 1755–1797. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.findings-naacl.96. URL https://aclanthology.org/2025.findings-naacl.96/.

Marc Lanctot, Vinicius Zambaldi, Audrunas Gruslys, Angeliki Lazaridou, Karl Tuyls, Julien Perolat, David Silver, and Thore Graepel. A Unified Game-Theoretic Approach to Multiagent Reinforcement Learning. *arXiv preprint arXiv:1711.00832*, 2017. URL https://arxiv.org/abs/1711.00832.

Yoonho Lee, Joseph Boen, and Chelsea Finn. Feedback descent: Open-ended text optimization via pairwise comparison. *arXiv preprint arXiv:2511.07919*, 2025. URL https://arxiv.org/abs/2511.07919.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. CAMEL: Communicative Agents for "Mind" Exploration of Large Language Model Society. *arXiv preprint arXiv:2303.17760*, 2023. URL https://arxiv.org/abs/2303.17760.

Lihong Li, Thomas J. Walsh, and Michael L. Littman. Towards a unified theory of state abstraction for MDPs. In *International Symposium on Artificial Intelligence and Mathematics*, 2006. URL https://thomasjwalsh.net/pub/aima06Towards.pdf.

Jonathan Light, Min Cai, Weiqin Chen, Guanzhi Wang, Xiusi Chen, Wei Cheng, Yisong Yue, and Ziniu Hu. Strategist: Self-improvement of LLM Decision Making via Bi-Level Tree Search. *arXiv preprint arXiv:2408.10635*, 2024. URL https://arxiv.org/abs/2408.10635.

Bo Liu, Leon Guertler, Simon Yu, Zichen Liu, Penghui Qi, Daniel Balcells, Mickel Liu, Cheston Tan, Weiyan Shi, Min Lin, Wee Sun Lee, and Natasha Jaques. SPIRAL: Self-play on zero-sum games incentivizes reasoning via multi-agent multi-turn reinforcement learning. *arXiv preprint arXiv:2506.24119*, 2025a. URL https://arxiv.org/abs/2506.24119.

Mickel Liu, Liwei Jiang, Yancheng Liang, Simon Shaolei Du, Yejin Choi, Tim Althoff, and Natasha Jaques. Chasing moving targets with online self-play reinforcement learning for safer language models. *arXiv preprint arXiv:2506.07468*, 2025b. URL https://arxiv.org/abs/2506.07468.

Tianci Liu, Ran Xu, Tony Yu, Ilgee Hong, Carl Yang, Tuo Zhao, and Haoyu Wang. OpenRubrics: Towards scalable synthetic rubric generation for reward modeling and LLM alignment. *arXiv preprint arXiv:2510.07743*, 2025c. URL https://arxiv.org/abs/2510.07743.

Simone Moran, Yoella Bereby-Meyer, and Max Bazerman. Stretching the effectiveness of analogical training in negotiations: Teaching diverse principles for creating value. *Negotiation and Conflict Management Research*, 1(2):99–134, 2008. URL https://ncmr.lps.library.cmu.edu/article/89/galley/92/view/.

Janice Nadler, Leigh Thompson, and Leaf Van Boven. Learning negotiation skills: Four models of knowledge creation and transfer. *Management Science*, 49(4):529–540, 2003. doi: 10.1287/mnsc.49.4.529.14431. URL https://doi.org/10.1287/mnsc.49.4.529.14431.

Krista Opsahl-Ong, Michael J Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. Optimizing instructions and demonstrations for multi-stage language model programs. *arXiv preprint arXiv:2406.11695*, 2024. URL https://arxiv.org/abs/2406.11695.

Martin J. Osborne and Ariel Rubinstein. *A Course in Game Theory*. MIT Press, 1994. URL https://books.osborne.economics.utoronto.ca/.

Reid Pryzant, Dan Iter, Jerry Li, Yin Lee, Chenguang Zhu, and Michael Zeng. Automatic Prompt Optimization with "Gradient Descent" and Beam Search. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 7957–7968. Association for Computational Linguistics, 2023. URL https://aclanthology.org/2023.emnlp-main.494/.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language Agents with Verbal Reinforcement Learning. *arXiv preprint arXiv:2303.11366*, 2023. URL https://arxiv.org/abs/2303.11366.

Yoav Shoham, Rob Powers, and Trond Grenager. If multi-agent learning is the answer, what is the question? *Artificial Intelligence*, 171(7):365–377, 2007. doi: 10.1016/j.artint.2006.02.006. URL https://doi.org/10.1016/j.artint.2006.02.006.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. *arXiv preprint arXiv:1712.01815*, 2017. URL https://arxiv.org/abs/1712.01815.

Ziyu Wan, Yunxiang Li, Xiaoyu Wen, Yan Song, Hanjing Wang, Linyi Yang, Mark Schmidt, Jun Wang, Weinan Zhang, Shuyue Hu, and Ying Wen. ReMA: Learning to meta-think for LLMs with multi-agent reinforcement learning. *arXiv preprint arXiv:2503.09501*, 2025. URL https://arxiv.org/abs/2503.09501.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation. *arXiv preprint arXiv:2308.08155*, 2023. URL https://arxiv.org/abs/2308.08155.

Yunfei Xie, Kevin Wang, Bobby Cheng, Jianzhu Yao, Zhizhou Sha, Alexander Duffy, Yihan Xi, Hongyuan Mei, Cheston Tan, Chen Wei, Pramod Viswanath, and Zhangyang Wang. MEMO: Memory-augmented model context optimization for robust multi-turn multi-agent LLM games. *arXiv preprint arXiv:2603.09022*, 2026. URL https://arxiv.org/abs/2603.09022.

Ran Xu, Tianci Liu, Zihan Dong, Tony Yu, Ilgee Hong, Carl Yang, Linjun Zhang, Tuo Zhao, and Haoyu Wang. Alternating reinforcement learning for rubric-based reward modeling in non-verifiable LLM post-training. *arXiv preprint arXiv:2602.01511*, 2026. URL https://arxiv.org/abs/2602.01511.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large Language Models as Optimizers. *arXiv preprint arXiv:2309.03409*, 2023. URL https://arxiv.org/abs/2309.03409.

Jianzhu Yao, Kevin Wang, Ryan Hsieh, Haisu Zhou, Tianqing Zou, Zerui Cheng, Zhangyang Wang, and Pramod Viswanath. SPIN-Bench: How well do LLMs plan strategically and reason socially? *arXiv preprint arXiv:2503.12349*, 2025. URL https://arxiv.org/abs/2503.12349.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. TextGrad: Automatic "differentiation" via text. *arXiv preprint arXiv:2406.07496*, 2024. URL https://arxiv.org/abs/2406.07496.

Wenqi Zhang, Ke Tang, Hai Wu, Mengna Wang, Yongliang Shen, Guiyang Hou, Zeqi Tan, Peng Li, Yueting Zhuang, and Weiming Lu. Agent-Pro: Learning to Evolve via Policy-Level Reflection and Optimization. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5348–5375. Association for Computational Linguistics, 2024. URL https://aclanthology.org/2024.acl-long.292/.

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, pp. 19632–19642, 2024. URL https://ojs.aaai.org/index.php/AAAI/article/view/29936.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large Language Models Are Human-Level Prompt Engineers. *arXiv preprint arXiv:2211.01910*, 2022. URL https://arxiv.org/abs/2211.01910.

# APPENDIX

## A RELATED WORK

**Multi-agent learning and self-play.** Self-play and multi-agent reinforcement learning (MARL) learn strategies through repeated interaction. AlphaZero learns through self-play in sequential games, while policy-space response oracles train best responses to mixtures of other players' strategies (Silver et al., 2017; Lanctot et al., 2017). For LLMs, SPIRAL uses competitive games to develop reasoning through self-play RL (Liu et al., 2025a). UnstableBaselines provides an online RL framework for training LLMs through self-play on text-based games (Guertler et al., 2025b). Practical applications include co-evolving attackers and defenders in Self-RedTeam (Liu et al., 2025b), alternating updates to a rubric generator and judge in Rubric-ARM (Xu et al., 2026), and cooperative planning and reasoning in ReMA (Wan et al., 2025). Language Self-Play also formulates language tasks as interactions between a challenger that generates requests and a solver that answers them (Kuba et al., 2025). These methods improve behavior through model weight updates. SPOON optimizes explicit textual strategies through interaction while keeping model weights fixed. For single-agent tasks, this interaction pairs the solver with an adaptive adversary.

**Prompt optimization.** Prompt optimization improves model behavior by searching over instructions while keeping model weights fixed. APE and OPRO use LLMs to generate and search over candidate instructions (Zhou et al., 2022; Yang et al., 2023). ProTeGi and TextGrad use textual feedback to guide revisions (Pryzant et al., 2023; Yuksekgonul et al., 2024), while MIPRO optimizes instructions and demonstrations within multi-stage programs (Opsahl-Ong et al., 2024). GEPA combines trajectory reflection with evolutionary prompt search (Agrawal et al., 2025), and Feedback Descent revises text using feedback from pairwise comparisons (Lee et al., 2025). Related methods such as Reflexion and ExpeL retain lessons from past interactions as textual guidance for future decisions (Shinn et al., 2023; Zhao et al., 2024).

**Prompt and context optimization for LLM games.** MEMO combines prompt search, persistent memory, and prioritized replay, evaluating candidate contexts against an opponent using a fixed default prompt (Xie et al., 2026). Agent-Pro learns behavioral guidelines and descriptions of opponents from failed games and verifies updates by replaying the same game (Zhang et al., 2024). Strategist uses population-based self-play to improve value heuristics for tree search and dialogue guides (Light et al., 2024). SPOON organizes strategies into explicit strategic states and behavioral rules. Following the motivation of state abstraction (Li et al., 2006), it groups histories by their shared strategic problem so that one rule can guide decisions across them. This representation lets the optimizer target changes to individual conditions or rules and select affected stored examples for verification. The optimizer also examines both players' strategies to guide structured best-response search as their strategies co-evolve.

## B LIMITATIONS

We evaluate SPOON's strategy optimization only on Qwen models with 7B or 8B parameters. Although our evaluation includes larger held-out opponents, we have not tested optimization of larger or frontier models. As a prompt optimizer, SPOON requires no access to model weights and can thus be applied to these models through their APIs. Our formulation and experiments also focus on two-player interactions. Although explicit textual strategies and best responses can be defined for games with more players, adapting and evaluating SPOON's optimization loop in these settings remains future work.