# Multi-Agent Reinforcement Learning in Non-Cooperative Stochastic Games Using Large Language Models

Shayan Meshkat Alsadat and Zhe Xu, *Member, IEEE*

***Abstract*—We study the use of large language models (LLMs) to integrate high-level knowledge in stochastic games using reinforcement learning with reward machines to encode non-Markovian and Markovian reward functions. In non-cooperative games, one challenge is to provide agents with knowledge about the task efficiently to speed up the convergence to an optimal policy. We aim to provide this knowledge in the form of deterministic finite automata (DFA) generated by LLMs (LLM-generated DFA). Additionally, we use reward machines (RMs) to encode the temporal structure of the game and the non-Markovian or Markovian reward functions. Our proposed algorithm, LLM-generated DFA for Multi-agent Reinforcement Learning with Reward Machines for Stochastic Games (StochQ-RM), can learn an equivalent reward machine to the ground truth reward machine (specified task) in the environment using the LLM-generated DFA. Additionally, we propose DFA-based q-learning with reward machines (DBQRM) to find the best responses for each agent using Nash equilibrium in stochastic games. Despite the fact that the LLMs are known to hallucinate, we show that our method is robust and guaranteed to converge to an optimal policy. Furthermore, we study the performance of our proposed method in three case studies.**

***Index Terms*—Reinforcement learning, large language models, stochastic games, reward machines.**

## I. Introduction

WE STUDY multi-agent reinforcement learning (MARL) for non-cooperative games with reward machines (RMs) using large language models (LLMs) where agents learn an *optimal policy* that maximizes their discounted cumulative reward. Agents' actions depend on each other since agents' strategies (policies) are not mutually exclusive. Hence, learning the *best response*, the response that no agent is motivated to deviate from unilaterally, i.e., Nash equilibrium, requires agents to learn the policy of their opponent.

StochQ-RM is a multi-agent reinforcement learning framework using LLM-generated DFAs (deterministic finite automata) for q-learning with RMs in stochastic games. This decentralized algorithm enables agents to learn optimal policies at Nash equilibrium through q-functions. RMs decompose tasks into sub-tasks, improving convergence to an optimal policy [1]. Unlike methods requiring human experts to provide domain-specific knowledge, StochQ-RM leverages LLMs to extract task-specific information. LLMs contain vast domain knowledge, reducing the need for human involvement [2].

In our method, generative language models (GLMs) are used to extract high-level knowledge for multi-agent reinforcement learning, aiming to learn an optimal policy efficiently. GLMs help obtain the LLM-generated DFA of each agent, which aids in generating the required task DFA. This knowledge facilitates learning an equivalent RM that captures the game's temporal structure. To minimize inefficiencies, we utilize prompting techniques such as chain-of-thought [3], zero-shot [4], and few-shot [5] methods to obtain the LLM-generated DFA of agents, avoiding the high resource demand of fine-tuning [6]. This approach minimizes LLM hallucinations [7] and ensures robust convergence to an optimal policy. Our algorithm employs the Lemke-Howson [8] method to learn q-functions at Nash equilibrium.

The baselines we compare are Nash-Q [9] (Nash-Q-learning) that uses Nash equilibrium to learn an optimal policy, Nash-QAS (Nash-Q-learning in augmented state space), MADDPG-SG (multi-agent deep deterministic policy gradient) [10] that is based on the actor-critic method, MADDPG-AS (extension of MADDPG to augmented state space), and QRM-SG (Q-learning for RM in stochastic games) [11]. Our contributions include: (a) Learning high-level tasks with temporal structures using LLM-generated DFA. (b) Guaranteeing convergence to the ground truth RM. (c) Ensuring robustness against LLM hallucinations and guaranteeing convergence to an optimal policy.

**Related works: MARL with RM.** In MARL, RMs improve q-learning efficiency to converge to an optimal policy [12]. Existing methods for learning ground truth RM are costly and fail in complex, non-stationary settings [13]. We instead leverage

Received 16 September 2024; revised 12 November 2024; accepted 27 November 2024. Date of publication 11 December 2024; date of current version 18 December 2024. This work was supported in part by the NSF under Grant CNS 2304863, Grant CNS 2339774, and Grant IIS 2332476; and in part by the ONR under Grant N00014-23-1-2505. Recommended by Senior Editor C. Briat. *(Corresponding author: Zhe Xu.)*

The authors are with the Faculty of Mechanical Engineering, Arizona State University, Tempe, AZ 85281 USA (e-mail: smeshka1@asu.edu; xzhe1@asu.edu).

Digital Object Identifier 10.1109/LCSYS.2024.3515879

LLM-generated DFA for a more efficient solution. **Language models and automata.** LLMs can generate instructions and simplify knowledge [14], and can produce automata from instructions (e.g., GLM2FSA) to model domain-specific tasks [15]. **Learning reward machines.** Researchers in [16] proposed AdvisoRL, which learns an RM using human expert advice. Our method provides a more efficient way to learn the ground truth RM than existing methods.

## II. Preliminaries

This section introduces the necessary background and notations for reinforcement learning (RL), deterministic finite automaton (DFA), and reward machines (RMs).

*Definition 1 (Stochastic Game):* We denote a labeled two-agent stochastic game similar to [11] by $\mathcal{G} = \langle S, s_I, A_\mathsf{e}, A_\mathsf{a}, \mathcal{R}_\mathsf{e}, \mathcal{R}_\mathsf{a}, p, \gamma, \mathcal{P}_\mathsf{e}, \mathcal{P}_\mathsf{a}, L_\mathsf{e}, L_\mathsf{a}\rangle$ where $S = S_\mathsf{e} \times S_\mathsf{a}$ is the finite set of states consisting of both agents' state $s = [s_\mathsf{e}, s_\mathsf{a}]$, $s_\mathsf{e} \in S_\mathsf{e}$ and $s_\mathsf{a} \in S_\mathsf{a}$ (states of both ego $\mathsf{e}$ and adverserial $\mathsf{a}$ agents). $A_\mathsf{e}$ and $A_\mathsf{a}$ are the finite set of actions for both ego and adversarial agents, $p = S \times A_\mathsf{e} \times A_\mathsf{a} \times S \to [0, 1]$ is the probabilistic transition function, $R_\mathsf{e} = (S \times A_\mathsf{e} \times A_\mathsf{a})^+ \times S \to \mathbb{R}$ ($+$ denotes the history of visited states and actions) and $R_\mathsf{a} = (S \times A_\mathsf{e} \times A_\mathsf{a})^+ \times S \to \mathbb{R}$ are the reward functions of ego and adversarial agents, respectively. $\gamma \in [0, 1]$ is the discount factor, $\mathcal{P}_\mathsf{e}$ and $\mathcal{P}_\mathsf{a}$ are finite sets of propositions, and $L_\mathsf{e} : S \times A_\mathsf{e} \times A_\mathsf{a} \times S \to 2^{\mathcal{P}_\mathsf{e}}$ and $L_\mathsf{a} : S \times A_\mathsf{e} \times A_\mathsf{a} \times S \to 2^{\mathcal{P}_\mathsf{a}}$ are the labeling functions of both ego and adverserial agents. We denote the size of a stochastic game as $|\mathcal{G}| = |S|$ where $|\cdot|$ is the set cardinality. $p(a \mid b)$ is the conditional probability.

The reward functions $R_\mathsf{e}$ and $R_\mathsf{a}$ are non-Markovian. We assume that each agent observes the trajectory of the other.

A *policy* (also referred to as strategy in game theory) is a function mapping from states to actions denoted by $\pi_i^j(s_{\mathsf{e},k}, s_{\mathsf{a},k}, a_{i,k})$ (policy of agent $i$ at time $k$ estimated by agent $j$) where $s_{\mathsf{e},k}, s_{\mathsf{a},k} \in S$, $a_{i,k} \in A$, and $i, j \in \{\mathsf{e}, \mathsf{a}\}$ at time $k \in \mathbb{N}$. It shows the probability of taking action $a_{i,k}$ at time $k$ in state $s_{i,k}$ with probability of $\pi_i^j : S \times A_i \to [0, 1]$ and moving to state $s_{i,k+1}$, where $s_{i,k+1}$ is the next state at time $k + 1$. For brevity, we denote $s = [s_\mathsf{e}, s_\mathsf{a}]$ and $s' = [s'_\mathsf{e}, s'_\mathsf{a}]$ such that $s_i = s_{i,k}$ and $s'_i = s_{i,k+1}$.

*Remark 1:* For any quantity with two subscripts, such as $s_{\mathsf{e},k}$, the first subscript is for the agent, and the second is for the time. In the case of only one subscript, the subscript refers to the agent.

A *trajectory* is a sequence of state-action pairs: $(s_\mathsf{e}, a_\mathsf{e}, s_\mathsf{a}, a_\mathsf{a}, s'_\mathsf{e}, s'_\mathsf{a})$, representing the realization of a stochastic process under policies $\pi_\mathsf{e}$ and $\pi_\mathsf{a}$. An optimal policy is denoted by $\pi_i^\star$. The corresponding label sequence for this process is $l_{i,0}l_{i,1}\ldots l_{i,n}$, obtained from $L_i$. A *trace* is defined as the pair of labels and rewards for the trajectory: $(\lambda_i, \rho_i) = (l_{i,0}l_{i,1}\ldots l_{i,n}, r_{i,0}r_{i,1}\ldots r_{i,n})$, with the discounted cumulative reward being $\sum_k \gamma^k r_{i,k}$. Each agent aims to maximize its expected discounted cumulative reward, i.e., $\mu_i(s_\mathsf{e}, \pi_\mathsf{e}, s_\mathsf{a}, \pi_\mathsf{a})$ using its learned policy $\pi_i$.

$$\mu_i(s_\mathsf{e}, \pi_\mathsf{e}, s_\mathsf{a}, \pi_\mathsf{a}) = \sum_{k=0}^{\infty} \gamma^k \mathbb{E}\big(r_{i,k} | \pi_\mathsf{e}, \pi_\mathsf{a}, s_{\mathsf{e},0} = s_\mathsf{e}, s_{\mathsf{a},0} = s_\mathsf{a}\big) \tag{1}$$

*Definition 2 (Nash Equilibrium of a Stochastic Game):* We define the Nash equilibrium of a stochastic game as the set of optimal policies that maximize $\mu_i$ such that

$$\mu_\mathsf{e}\big(s_\mathsf{e}, \pi_\mathsf{e}^\star, s_\mathsf{a}, \pi_\mathsf{a}^\star\big) \geq \mu_\mathsf{e}\big(s_\mathsf{a}, \pi_\mathsf{e}, s_\mathsf{e}, \pi_\mathsf{a}^\star\big) \;\forall\; \pi_\mathsf{e} \tag{2}$$

$$\mu_\mathsf{a}\big(s_\mathsf{e}, \pi_\mathsf{e}^\star, s_\mathsf{a}, \pi_\mathsf{a}^\star\big) \geq \mu_\mathsf{a}\big(s_\mathsf{a}, \pi_\mathsf{e}^\star, s_\mathsf{e}, \pi_\mathsf{a}\big) \;\forall\; \pi_\mathsf{a} \tag{3}$$

*Definition 3 (Deterministic Finite Automaton):* A DFA is a finite state automaton (FSA) whose states are finite sets and whose transitions are labeled with finite sets. We denote a DFA by a tuple $\mathscr{H} = \langle H, h_I, \Sigma, \delta, F\rangle$ where $H$ is the set of states, $h_I$ is the initial state, $\Sigma$ is the input alphabet, $\delta : H \times \Sigma \to H$ is the transition function, and $F \subseteq H$ is the set of accepting states. Let the size of the DFA $\mathscr{H}$ be $|H|$.

*Definition 4 (Reward Machine):* We denote a reward machine by $\mathcal{A} = \langle V, v_I, 2^\mathcal{P}, M, \Gamma, \sigma\rangle$ where $V$ is a finite set of states, $v_I$ is the initial state, $2^\mathcal{P}$ is the input alphabet, $M \subseteq \mathbb{R}$ is output alphabet, $\Gamma : V \times 2^\mathcal{P} \to V$ is the set transition function, and $\sigma : V \times 2^\mathcal{P} \to M$ is the output function. We define the size of the reward machine as $|\mathcal{A}| = |V|$.

By applying the reward machine $\mathcal{A}^i$ on a sequence of labels $l_{i,0}l_{i,1}\ldots l_{i,n}$, we obtain a sequence of RM states $v_{i,0}(l_{i,0}, r_{i,0})v_{i,1}(l_{i,1}, r_{i,1})\ldots v_{i,n}(l_{i,n}, r_{i,n})$ where $v_{i,0} = v_{i,I}$ is the initial state of the RM.

## III. Stochastic Games

We study two-player zero-sum and general-sum stochastic games. Our methodology can be extended to more than two agents by using methods that find meta Nash equilibrium (e.g., [17]) instead of Lemke-Howson. For StochQ-RM, we propose DFA-based q-learning with RMs, DBQRM, a decentralized algorithm where each agent learns its policy as it interacts with the adversary.

Our algorithm finds the Nash equilibrium at each time step (best response to the opponent's action). This action transitions the agent to the next augmented state and yields a reward, which is used to update the policy. Through this interaction with the environment and adversarial agent, we derive an optimal policy that maximizes each agent's discounted cumulative reward (the existence of Nash equilibria for stochastic games has been studied [18]).

*Definition 5 (q-Function at a Nash Equilibrium):* We define the q-function at a Nash equilibrium for each agent as:

$$\begin{aligned} q_i^j(s, v_\mathsf{e}, v_\mathsf{a}, a_\mathsf{e}, a_\mathsf{a}) &= r_i(s, v_\mathsf{e}, v_\mathsf{a}, a_\mathsf{e}, a_\mathsf{a}) \\ &+\gamma \sum_{s' \in S; v'_\mathsf{e} \in V_\mathsf{e}; v'_\mathsf{a} \in V_\mathsf{a}} p\big(s', v'_\mathsf{e}, v'_\mathsf{a} | s, v'_\mathsf{e}, v'_\mathsf{a}, a_\mathsf{e}, a_\mathsf{a}\big)\mu_i\big(s', v'_\mathsf{e}, v'_\mathsf{a}, \pi_\mathsf{e}^\star, \pi_\mathsf{a}^\star\big) \end{aligned} \tag{4}$$

The total discounted reward for agent $i$ is given by $\mu_i(s', v'_\mathsf{e}, v'_\mathsf{a}, \pi_\mathsf{e}^\star, \pi_\mathsf{a}^\star)$, based on the augmented state $(s', v'_\mathsf{e}, v'_\mathsf{a})$. Each agent maintains its own q-function and the opponent's estimation, denoted as $q_i^j$ (e.g., $q_i^\mathsf{e}, q_\mathsf{a}^\mathsf{e}$). The Nash equilibrium at each time step to the stochastic game is computed using the Lemke-Howson algorithm [8], determining each agent's equilibrium action, i.e., the best response to the opponent's actions. This action transitions the agent to the next $\mathcal{G}$ state $s'$ and yields a high-level event label $l_k$, leading to a transition to the RM state $v'_i$ and the corresponding reward $r_i$.

The agents then update their q-values based on the Nash equilibrium.

$$\bar{q}_i^j(s', v'_{\mathrm{e}}, v'_{\mathrm{a}}) = \pi_{\mathrm{e}}^j(s', v'_{\mathrm{e}}, v'_{\mathrm{a}}) \otimes \pi_{\mathrm{a}}^j(s', v'_{\mathrm{e}}, v'_{\mathrm{a}}) \cdot q_i^j(s', v'_{\mathrm{e}}, v'_{\mathrm{a}}) \tag{5}$$

$$q_i^j(s, v_{\mathrm{e}}, v_{\mathrm{a}}, a_{\mathrm{e}}, a_{\mathrm{a}}) = (1-\alpha) q_i^j(s, v_{\mathrm{e}}, v_{\mathrm{a}}, a_{\mathrm{e}}, a_{\mathrm{a}}) + \alpha\Big(r_i + \gamma \bar{q}_i^j(s', v'_{\mathrm{e}}, v'_{\mathrm{a}})\Big) \tag{6}$$

where the $\pi_{\mathrm{e}}^j(s', v'_{\mathrm{e}}, v'_{\mathrm{a}})$ and $\pi_{\mathrm{a}}^j(s', v'_{\mathrm{e}}, v'_{\mathrm{a}})$ denote the Nash equilibrium solution to the stage game and $\pi_{\mathrm{e}}^j \otimes \pi_{\mathrm{a}}^j$ ($\otimes$ denotes the Kronecker product) shows the joint probabilities of actions. $\bar{q}_i^j$ is the best response to the opponent's action.

## IV. Extraction of Domain-Specific Knowledge From Large Language Models

We use a combination of persona adoption, zero-shot, and few-shot methods that we call *MixedR* (mixed-reasoning) to obtain the LLM-generated DFA for each agent.

*Example 1:* In a variation of a Pac-Man game, agents need to reach their own powerhouse to recharge, and then they must reach their adversary's powerhouse to dominate it. Afterward, they can conquer their adversary's base (see Figure 4). We describe this task as *task description* to LLM using prompt engineering to obtain the LLM-generated DFA.

In Example 1, we use the persona of a zero-sum game expert to guide the LLM in generating coherent outputs by focusing on a domain and providing a task description for DFA generation to minimize hallucinations. MixedR in Example 1 yields LLM-generated DFA $\mathscr{H}_i^j$ for each agent.

*Prompt Example 1:* You are an expert in RL for "zero-sum" games. Provide a strategy for each agent in "task description" (Example 1). Provide your answer in the form of deterministic finite automata (DFA). Here are some examples of DFA. The format is ⟨task desctiption⟩ − ⟨DFA⟩.

- ⟨**task description No.1**⟩ − ⟨**DFA No.1**⟩
- ...

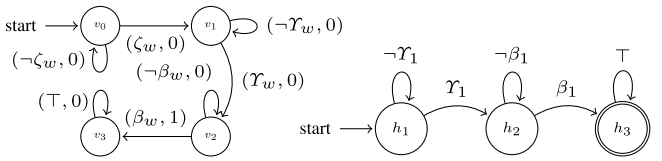
Let us think step by step.

Prompt Example 1 shows an example of MixedR.

## V. Using LLMs to Generate DFA

In this setup, DFA $\mathscr{H}$ operates on a sequence of labels $\lambda = l_1 l_2 \dots l_n \in \Sigma^\star$, which maps to a sequence of DFA states $h_0 h_1 \dots h_n$, with transitions defined as $h_{k+1} = \delta(h_k, l_k)$. The formal language accepted by $\mathscr{H}$ is defined as $\mathcal{L}(\mathscr{H}) = l_1 l_2 \dots l_n \in \Sigma^\star | h_n \in F$. An LLM-generated DFA yielding a positive reward is considered a *compatible* DFA, where the label sequence $\lambda_i^j \in \mathcal{L}(\mathscr{H}_i^j)$ indicates potential reward. This approach narrows the search space for learning the ground truth RM, and StochQ-RM disregards any hallucinated LLM-generated DFA that is incompatible with the ground truth RM, $\mathcal{A}^j$ (i.e., $\mathcal{A}^{\mathrm{e}}$ and $\mathcal{A}^{\mathrm{a}}$).

*Definition 6 (Compatibility of LLM-Generated DFA):* We consider an LLM-generated DFA $\mathscr{H}_i^j$ to be compatible with RM $\mathcal{A}^j$ if returning a positive reward, i.e., $r_k > 0$ implies $l_1 l_2 \dots l_k \in \mathcal{L}(\mathscr{H}_i^j)$ for all label sequences $l_1 l_2 \dots l_k \in (2^{\mathcal{P}})^+$ with the corresponding reward sequence of $\mathcal{A}^j(l_1 l_2 \dots l_k) = r_1 r_2 \dots r_k$ for $k \in \mathbb{N}$.




(a) Ground truth RM (b) LLM-generated DFA $\mathscr{H}$

Fig. 1. Compatible LLM-generated DFA $\mathscr{H}$ is used to learn the ground truth RM. Note that DFA states are denoted by $h$ and RM states are denoted by $v$, and $w \in \{1, 2\}$.

**Algorithm 1:** `PromptLLMforDFA`: Constructing Task DFA for a Domain-Specific Task Using LLM

**Input**: prompt $f$
**Parameter**: temperature $T$, Top p $P$ (OpenAI LLM parameters)
**Output**: DFA

1 *output* ← `PromptLLM`($f, T, P$);
2 $H \leftarrow \{h_0\}$, $h_0 \leftarrow 0$, $\delta \leftarrow \{\varpi_0\}$, $\mathcal{P} \leftarrow \{\mathscr{P}_0\}$, $\mathcal{K} \leftarrow 0$;
3 **for** *instruction* ∈ *output* **do**
4 $\quad \mathcal{K} \leftarrow \mathcal{K} + 1$;
5 $\quad h_{\mathcal{K}} \leftarrow$ `GetStates`(*instruction*);
6 $\quad \mathscr{P}_{\mathcal{K}} \leftarrow$ `GetProposition`(*instruction*);
7 $\quad \varpi_{\mathcal{K}} \leftarrow$ `ComputeTransition`($h_{\mathcal{K}}, \mathscr{P}_{\mathcal{K}}$);
8 $\quad H \leftarrow H \cup \{h_{\mathcal{K}}\}$, $\delta \leftarrow \delta \cup \{\varpi_{\mathcal{K}}\}$, $\mathcal{P} \leftarrow \mathcal{P} \cup \{\mathscr{P}_{\mathcal{K}}\}$;
9 **return** $\mathscr{H} = \langle H, h_0, 2^{\mathcal{P}}, \delta, \{h_{\mathcal{K}}\}\rangle$;

*Example 2:* Consider ground truth RM in Figure 1(a) and the compatible LLM-generated DFA $\mathscr{H}$ in Figure 1(b) where an agent can use $\mathscr{H}$ to learn the ground truth RM which may lead to faster convergence to an optimal policy.

If the LLM-generated DFA is incompatible with the ground truth RM, i.e., a *counterexample*, StochQ-RM adjusts the prompt using the counterexample to obtain a more compatible LLM-generated DFA. Counterexample occurs when a label sequence and its corresponding reward sequence that is suggested by the LLM-generated DFA is inconsistent with the ground truth. We use Algorithm 1 to obtain the LLM-generated DFA. We create a prompt $f$ using MixedR, providing instructions to generate the DFA $\mathscr{H}_i^j$. MixedR ensures the LLM generates output in a structured format (Line 1). Next, we define the DFA's set of states $H$, transitions $\delta$, and propositions $\mathcal{P}$ (Line 2). For each LLM output (the DFA), we extract the states, transitions, and the propositions triggering those transitions (Lines 3-7). This process is repeated for all instructions to construct the domain-specific LLM-generated DFA for agent $i$ (Line 9).

We apply DBQRM (Algorithm 2) to find the Nash equilibrium using an extended Lemke-Howson method with a lexicographic approach [8]. After initializing agent and RM states (Lines 1–2), we create empty label and reward sequence sets (Line 3). At each time step $t$ both agents share their augmented state and action with their adversary, meaning that the agents have access to the opponent's trajectory. We obtain the Nash equilibrium action as the best response by estimating the opponent's policy, e.g., the ego agent estimates

**Algorithm 2:** DBQRM

**Hyperparameters**: episode length $ep_{\texttt{len}}$, learning rate $\alpha$, discount factor $\gamma$, epsilon-greedy $\varepsilon$

**Input**: RMs $\mathcal{A}^{\texttt{e}}, \mathcal{A}^{\texttt{a}}, \mathcal{U}^{\texttt{e}}_{\texttt{e}}, \mathcal{U}^{\texttt{e}}_{\texttt{a}}, \mathcal{U}^{\texttt{a}}_{\texttt{e}}, \mathcal{U}^{\texttt{a}}_{\texttt{a}}$, and Q-functions $\left(q^{\texttt{e}}_{\texttt{e}}, q^{\texttt{a}}_{\texttt{e}}, q^{\texttt{e}}_{\texttt{a}}, q^{\texttt{a}}_{\texttt{a}}\right)$, temperature $T$, Top p $P$, prompt $f$

1 $s \leftarrow$ `InitialState()`;
2 $v_{\texttt{e}} \leftarrow v_{I,\texttt{e}}$, $v_{\texttt{a}} \leftarrow v_{I,\texttt{a}}$, $v_{\mathcal{A}^{\texttt{e}}} \leftarrow v_{I,\mathcal{A}^{\texttt{e}}}$, $v_{\mathcal{A}^{\texttt{a}}} \leftarrow v_{I,\mathcal{A}^{\texttt{a}}}$;
3 $\lambda_{\texttt{e}} \leftarrow \{\}$, $\lambda_{\texttt{a}} \leftarrow \{\}$, $\rho_{\texttt{e}} \leftarrow \{\}$, $\rho_{\texttt{a}} \leftarrow \{\}$;
4 **for** $0 \le t < ep_{len}$ **do**
5  **for** $i \in \{\texttt{e}, \texttt{a}\}$ **do**
6  $\pi^i_{\texttt{e}}(\cdot|s, v_{\texttt{e}}, v_{\texttt{a}}), \pi^i_{\texttt{a}}(\cdot|s, v_{\texttt{e}}, v_{\texttt{a}}) \leftarrow$ `FindNashEq`$\left(q^i_{\texttt{e}}, q^i_{\texttt{a}}\right)$;
7  $a_i \leftarrow$ `SelectAction`$\left(\pi^i_i(a|s, v_{\texttt{e}}, v_{\texttt{a}}), \epsilon\right)$;
8  $s' \leftarrow$ `ExecuteAction`$(s, a_{\texttt{e}}, a_{\texttt{a}})$;
9  **for** $i \in \{\texttt{e}, \texttt{a}\}$ **do**
10  $l_{i,t} \leftarrow L_i(s, a_{\texttt{e}}, a_{\texttt{a}}, s')$;
11  $v'_{\mathcal{A}^i} \leftarrow \Gamma_{\mathcal{A}^i}(v_{\mathcal{A}^i}, l_{i,t})$, $r_{i,t} \leftarrow \delta_{\mathcal{A}^i}(v_{\mathcal{A}^i}, l_{i,t})$;
12  $v'_i \leftarrow \Gamma_i(v_i, l_{i,t})$;
13  **for** $i \in \{\texttt{e}, \texttt{a}\}$ **do**
14  $\pi^i_{\texttt{e}}(\cdot|s, v_{\texttt{e}}, v_{\texttt{a}}), \pi^i_{\texttt{a}}(\cdot|s, v_{\texttt{e}}, v_{\texttt{a}}) \leftarrow$ `FindNashEq`$\left(q^i_{\texttt{e}}(s, v_{\texttt{e}}, v_{\texttt{a}}), q^i_{\texttt{a}}(s, v_{\texttt{e}}, v_{\texttt{a}})\right)$;
15  $q^i_{\texttt{e}}(s, v_{\texttt{e}}, v_{\texttt{a}}), q^i_{\texttt{a}}(s, v_{\texttt{e}}, v_{\texttt{a}}) \leftarrow$ Equation (6);
16  $s \leftarrow s'$, $v_{\texttt{e}} \leftarrow v'_{\texttt{e}}$, $v_{\texttt{a}} \leftarrow v'_{\texttt{a}}$, $\lambda_i \leftarrow \lambda_i \cup \{l_{i,t}\}$, $\rho_i \leftarrow \rho_i \cup \{r_{i,t}\}$;
17 **return** $\left(q^{\texttt{e}}_{\texttt{e}}, q^{\texttt{a}}_{\texttt{e}}, q^{\texttt{a}}_{\texttt{a}}, \rho_{\texttt{e}}, \rho_{\texttt{a}}, \lambda_{\texttt{e}}, \lambda_{\texttt{a}}\right)$

**Algorithm 3:** StochQ-RM

**Hyperparameters**: $ep_{\texttt{len}}, \alpha, \gamma, \varepsilon$

**Input**: Ground truth RMs $\mathcal{A}^{\texttt{e}}, \mathcal{A}^{\texttt{a}}$, temperature $T$, Top p $P$, prompt $f$, query budgets $\mathcal{J}^{\texttt{e}}_{\texttt{e}}, \mathcal{J}^{\texttt{a}}_{\texttt{e}}, \mathcal{J}^{\texttt{e}}_{\texttt{a}}, \mathcal{J}^{\texttt{a}}_{\texttt{a}}$

1 $q^{\texttt{e}}_{\texttt{e}}, q^{\texttt{a}}_{\texttt{e}}, q^{\texttt{e}}_{\texttt{a}}, q^{\texttt{a}}_{\texttt{a}} \leftarrow$ `InitializeQfunction()`;
2 $X^{\texttt{e}}_{\texttt{e}}, X^{\texttt{a}}_{\texttt{e}}, X^{\texttt{e}}_{\texttt{a}}, X^{\texttt{a}}_{\texttt{a}} \leftarrow \emptyset$;
3 $f \leftarrow \{f_{\texttt{e}}, f_{\texttt{a}}\}$;
4 $\mathscr{H}^{\texttt{e}}_{\texttt{e}}, \mathscr{H}^{\texttt{a}}_{\texttt{e}}, \mathscr{H}^{\texttt{e}}_{\texttt{a}}, \mathscr{H}^{\texttt{a}}_{\texttt{a}} \leftarrow \{\}$, $\mathcal{U}^{\texttt{e}}_{\texttt{e}}, \mathcal{U}^{\texttt{a}}_{\texttt{e}}, \mathcal{U}^{\texttt{e}}_{\texttt{a}}, \mathcal{U}^{\texttt{a}}_{\texttt{a}} \leftarrow \{\}$;
5 **for** $j \in \{\texttt{e}, \texttt{a}\}$ **do**
6  **for** $i \in \{\texttt{e}, \texttt{a}\}$ **do**
7  $\mathscr{H}^j_i \leftarrow$ `PromptLLMforDFA`$(f_i, T, P)$;
8  $\mathcal{U}^j_i \leftarrow$ `InitializeRewardMachine`$\left(\mathscr{H}^j_i\right)$;
9 **for** *episode* $n = 1, 2, \ldots$ **do**
10  $X^{\texttt{e}}_{\texttt{e,init}}, X^{\texttt{a}}_{\texttt{e,init}}, X^{\texttt{e}}_{\texttt{a,init}}, X^{\texttt{a}}_{\texttt{a,init}} \leftarrow X^{\texttt{e}}_{\texttt{e}}, X^{\texttt{a}}_{\texttt{e}}, X^{\texttt{e}}_{\texttt{a}}, X^{\texttt{a}}_{\texttt{a}}$;
11  $(q^{\texttt{e}}_{\texttt{e}}, q^{\texttt{a}}_{\texttt{e}}, q^{\texttt{e}}_{\texttt{a}}, q^{\texttt{a}}_{\texttt{a}}, \rho_{\texttt{e}}, \rho_{\texttt{a}}, \lambda_{\texttt{e}}, \lambda_{\texttt{a}}) \leftarrow$ DBQRM$(\mathcal{A}^{\texttt{e}}, \mathcal{A}^{\texttt{a}}, \mathcal{U}^{\texttt{e}}_{\texttt{e}}, \mathcal{U}^{\texttt{a}}_{\texttt{e}}, \mathcal{U}^{\texttt{e}}_{\texttt{a}}, \mathcal{U}^{\texttt{a}}_{\texttt{a}}, q^{\texttt{e}}_{\texttt{e}}, q^{\texttt{a}}_{\texttt{e}}, q^{\texttt{e}}_{\texttt{a}}, q^{\texttt{a}}_{\texttt{a}})$;
12  **for** $j \in \{\texttt{e}, \texttt{a}\}$ **do**
13  **for** $i \in \{\texttt{e}, \texttt{a}\}$ **do**
14  **if** $\mathcal{U}^j_i(\lambda_i) \neq \rho_i$ **then**
15  $X^j_i \leftarrow X^j_i \cup \{(\lambda_i, \rho_i)\}$;
16  **if** $X^j_i \neq X^j_{i,init}$ **then**
17  **if** $\mathcal{J}^j_i \ge 0$ ***and*** $\exists(\lambda_i, \rho_i) \in X^j_i$, $\rho_i \ge 0$ ***and*** $\lambda_i \notin \mathcal{L}(\mathscr{H}^j_i)$ **then**
18  $f_i \leftarrow$ `UpdatePromptCntExmp`$\left(f_i, \lambda_j\right)$;
19  $\mathscr{H}^j_i \leftarrow$ `PromptLLMforDFA`$(f_i, T, P)$;
20  $\mathcal{J}^j_i \leftarrow \mathcal{J}^j_i - 1$;
21  $\mathcal{U}^j_i \leftarrow$ `LearnRewardMachine`$\left(\mathscr{H}^j_i, X^j_i\right)$;
22  $q^j_i \leftarrow$ `InitializeQfunction()`;

the adversary's policy $\pi^{\texttt{e}}_{\texttt{a}}$. By estimating the adversary's policy we obtain the actions $a_{\texttt{e}}$ and $a_{\texttt{a}}$ (Lines 5 - 7) for each agent. By executing the action we obtain the next state $s'$ and for each transition from state $s$ to $s'$ we obtain the label $l_{i,t}$, reward $r_{i,t}$ (given by the ground truth RM, Line 11), and the next RM state $v'_i$ (Lines 8–12). We then use the Lemke-Howson method to calculate the Nash equilibrium policy solution to the stage game (Line 14). DBQRM updates the q-values, $\mathcal{G}$ states, RM states and obtains the label sequences $\lambda_{\texttt{e}}$, $\lambda_{\texttt{a}}$ and reward sequences $\rho_{\texttt{e}}$, $\rho_{\texttt{a}}$ for each agent which later on will be used to find the ground truth RM (Lines 15–17).

StochQ-RM (Algorithm 3) learns RMs equivalent to the ground truth for the ego agent ($\mathcal{U}^{\texttt{e}}_{\texttt{e}}$, $\mathcal{U}^{\texttt{e}}_{\texttt{a}}$) and the adversary ($\mathcal{U}^{\texttt{a}}_{\texttt{e}}$, $\mathcal{U}^{\texttt{a}}_{\texttt{a}}$) by utilizing the LLM-generated DFA obtained by Algorithm 1 and updates optimal policies $\pi^\star_{\texttt{e}}$ and $\pi^\star_{\texttt{a}}$ using Algorithm 2. Initially, q-values $q^{\texttt{e}}_{\texttt{e}}, q^{\texttt{a}}_{\texttt{e}}, q^{\texttt{e}}_{\texttt{a}}, q^{\texttt{a}}_{\texttt{a}}$ and counterexample sets $X^{\texttt{e}}_{\texttt{e}}, X^{\texttt{a}}_{\texttt{e}}, X^{\texttt{e}}_{\texttt{a}}, X^{\texttt{a}}_{\texttt{a}}$ are initialized (Lines 1 - 2).

Prompts $f_{\texttt{e}}$ and $f_{\texttt{a}}$ are stored and their DFA sets initialized (Lines 3–4). The LLM is then prompted to generate the LLM-generated DFA (Lines 5, 8). Next, StochQ-RM uses DBQRM to find best-response policies (Line 11), and checks if the learned RM $\mathcal{U}^j_i$ matches the ground truth RM (Line 14). If the reward sequence is inconsistent with $\mathcal{G}$ (ground truth), the counterexample $(\lambda_i, \rho_i)$ is added to the counterexample set (Line 15). The algorithm verifies whether the counterexample is due to an incompatible LLM-generated DFA or the learned RM (Line 16). If the counterexample is due to the LLM-generated DFA (Line 17), the DFA is updated with the counterexample, and a new LLM-generated DFA is generated using Algorithm 1 (Lines 18 - 20). To manage LLM hallucinations, each agent is assigned a query budget $\mathcal{J}^j_i$. Based on our experiments, we have seen that for RMs with 5 to 7 states, LLM hallucinations frequency is once every 7 to 8 queries (on GPT4o) thus, we have chosen $\mathcal{J}^j_i = 20$ to be conservative. We have also observed that a low value of $\mathcal{J}^j_i = 3$ might be depleted in rare cases. The budget depends on the prompt; without a suitable prompt, we may not obtain the LLM-generated DFA, regardless of $\mathcal{J}^j_i$. If the budget is depleted without a correct LLM-generated DFA, we switch to learning from traces only (Line 20). We then refine $\mathcal{U}^j_i$ using the learned LLM-generated DFA and counterexamples (Line 21), and re-initialize q-values (Line 22).

## VI. Learning RM in Stochastic Games Using LLM-Generated DFA

Agents learn an equivalent RM to the ground truth RM by interacting with the environment. Each agent shares its observations with its adversary, ensuring the derived RM by the ego agent ($\mathcal{U}_{\mathtt{a}}^{\mathtt{e}}$) matches that of the adversary ($\mathcal{U}_{\mathtt{a}}^{\mathtt{a}}$), and vice versa. This equivalence holds since the counterexample $X_{\mathtt{a}}^{\mathtt{e}} \subseteq (2^{\mathcal{P}_i})^+ \times \mathbb{R}_i^+$ for $i \in \{\mathtt{e}, \mathtt{a}\}$ is the same for both agents. Throughout learning, each agent's opponent estimation remains consistent with the ground truth through shared observations. The learned RM $\mathcal{U}_i^j$ is consistent with the counterexample $X_i^j$, meaning $\mathcal{U}_i^j(\lambda_i) = \rho_i$ for all $(\lambda_i, \rho_i) \in X_i^j$. Using these counterexamples and label sequences, we learn an equivalent RM [13]. This learning iteratively generates propositional logic formulas $\phi_\kappa^{X_i^j}$, where $\kappa \in \mathbb{N}$, from the proposition set $\mathcal{P} = \{x, y, \ldots\}$ using Boolean connectives $\{\neg, \vee, \wedge, \mapsto\}$. The mapping $\mathcal{I} : \mathcal{P} \mapsto \{0, 1\}$ assigns Boolean values, with $\mathcal{I} \models \phi_\kappa^{X_i^j}$ indicating formula satisfaction. We increase the formula size until finding a satisfiable formula, representing an RM with $\kappa$ states consistent with the counterexamples.

## VII. Convergence to Optimal Policy

The learned RM $\mathcal{U}_i^j$ is considered equivalent to the ground truth RM $\mathcal{A}^j$ if it correctly recognizes any label sequence that is admissible within the underlying system $\mathcal{G}$ [16]. Given that LLMs may hallucinate, we must account for potential incompatibility in $\mathscr{H}$. To ensure that StochQ-RM converges correctly, we treat LLMs as potentially adversarial, meaning the worst-case scenario.

*Lemma 1:* Let $\mathcal{G}$ be a labeled stochastic game, $\mathcal{A}^j$ the ground truth RM encoding the rewards of $\mathcal{G}$, and $\mathscr{D}_i^j = \{\mathscr{H}_{i_1}^j, \ldots, \mathscr{H}_{i_m}^j\}$ the set of all LLM-generated DFAs ($|\mathscr{D}_i^j| = m \in \mathbb{N}$) that are added to $\mathscr{H}$ during the run of StochQ-RM for each agent. Additionally, let

$$n_{\max} = \max_{\mathscr{H}_i^j \in \mathscr{D}_i^j, z \in \{1,\ldots,m\}} \left\{ |\mathscr{H}_{i_z}^j| \right\} \tag{7}$$

$$\eta = \max_{i,j \in \{\mathtt{e},\mathtt{a}\}} \left\{ 2|\mathcal{G}| \cdot \left(|\mathcal{A}^j| + 1\right) \cdot n_{\max}, |\mathcal{G}|\left(|\mathcal{A}^j| + 1\right)^2 \right\} \tag{8}$$

Then, StochQ-RM with $ep_{\mathtt{len}} \geq \eta$ *almost surely* (meaning with probability 1) learns RM $\mathcal{U}_i^j$ that is equivalent to $\mathcal{A}^j$.

*Proof:* By showing that the set $\mathscr{D}_i^j$ stabilizes within a certain time interval, meaning no additional LLM-generated DFAs are introduced or removed, given the assumption that the agent encounters all possible $(s, a, l)$ infinitely often we can complete the proof. Here is the reasoning: if an LLM-generated DFA added to $\mathscr{D}_i^j$ is compatible, it remains in the set indefinitely. This is because no counterexample will invalidate it, preventing the condition in Line 16 from being met. Conversely, if an LLM-generated DFA is incompatible, the algorithm will eventually detect a trajectory that demonstrates this inconsistency, leading to its removal from $\mathscr{D}_i^j$. Additionally, the algorithm decreases $\mathcal{J}_i^j$ by one each time it adds a new LLM-generated DFA to $\mathscr{D}_i^j$, and this process continues only while $\mathcal{J}_i^j > 0$. Thus, the total number of LLM-generated DFAs created throughout the algorithm's execution is limited by $\mathcal{J}_i^j$. As a result, after a certain time interval, the set $\mathscr{D}_i^j$ stops changing since the algorithm either converges on the correct RM $\mathcal{A}^j$ or all incompatible DFAs have been removed, and LLM cannot be prompted ($\mathcal{J}_i^j = 0$).

Once $\mathscr{D}_i^j$ becomes fixed, it can be shown, similarly to Neider et al.'s proof [16], that the algorithm will eventually learn the true RM, i.e., $\mathcal{A}^j$. Also if the learned RM $\mathcal{A}^j$ is not equivalent to the ground truth RM $\mathcal{U}_i^j$, then the algorithm will collect a counterexample through its encounter with a trajectory that is incompatible with $\mathcal{U}_i^j$. Additionally, in the worst-case scenario, all trajectories of length $ep_{\mathtt{len}}$ will be incorporated into $X_i^j$, guaranteeing that the StochQ-RM will learn the correct RM. ■

StochQ-RM provides an upper bound for the episode length that needs to be explored. Its correctness follows the Lemma 1 and the correctness of the QRM algorithm [1].

*Theorem 1:* Let $\mathcal{G}$, $\mathcal{A}^j$, $\mathscr{D}_i^j$, $\eta$, and $m$ be as in Lemma 1. Then, StochQ-RM will converge to an optimal policy almost surely (meaning with probability 1) if $ep_{\mathtt{len}} \geq \eta$.

*Proof:* Theorem 1 follows from Lemma 1 and correctness of QRM algorithm [1]. ■

Theorem 1 guarantees the convergence of the StochQ-RM to an optimal policy if sufficient episode length is given for exploration. It also provides an upper bound for convergence, as illustrated in Lemma 1.

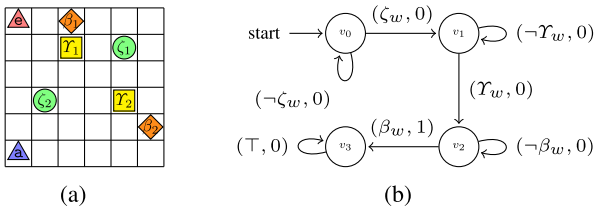
## VIII. Results

We study two scenarios: a zero-sum game (Pacman) and a general-sum game (Factory), investigating the performance of the StochQ-RM for each. We use `GPT4o` to obtain the LLM-generated DFA. In our case studies, tasks are encoded using RMs, and the *task description* is used to derive the LLM-generated DFA. We compare our method with **Nash-Q**, **Nash-QAS**, **MADDPG**, **MADDPG-AS**, and **QRM-SG**.

**Case Study 1.** In the general-sum factory game, each agent must complete a series of sub-tasks shown in Figure 2(b) within the environment (Figure 2(a)). We obtain the LLM-generated DFA, $\{\zeta_w, \Upsilon_w\}$ for agent $\mathtt{e}$ (Agent 1 in Figure 3) and agent $\mathtt{a}$ (Agent 2 in Figure 3), meaning that DFA reaches its accepting state when labels $\zeta_w$ and $\Upsilon_w$ are encountered. Figure 3 shows that the StochQ-RM converges to an optimal policy faster by using LLM-generated DFA $\mathscr{H}$, NashQAS can only converge to an optimal policy faster for Agent 1, and it is not as stable as StochQ-RM (see shaded area). Simulation results are for four independent runs averaged over each 200 episodes.

**Case Study 2.** We study StochQ-RM performance in zero-sum games. We consider the Pacman game similar to [11]. The obtained LLM-generated DFAs are $\{\langle \mathtt{e}, P_{\mathtt{e}} \rangle, \langle \mathtt{e}, P_{\mathtt{a}} \rangle\}$ and $\{\langle \mathtt{a}, P_{\mathtt{a}} \rangle, \langle \mathtt{a}, P_{\mathtt{e}} \rangle\}$ for agents $\mathtt{e}$ and $\mathtt{a}$, i.e., Agent 1 and 2 in Figure 5, respectively. Figure 5 shows that the StochQ-RM converges to an optimal policy faster by using LLM-generated DFA $\mathscr{H}$, simulation results are for five independent runs averaged over each 200 episodes.

**Case Study 3.** We test StochQ-RM in a different scenario of the factory game with grid dimension increased to $12 \times 12$, where the agents must visit a remote station located at




(a) (b)

Fig. 2. (a) Environment of general-sum game. (b) Ground truth RM of the general-sum game. $\beta_w$, $\Upsilon_w$ and $\zeta_w$ where $w \in \{1, 2\}$ are the tool stations, component locations, and assembly points, respectively, and triangles are the agents.

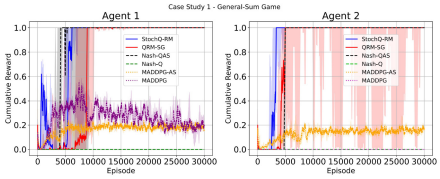
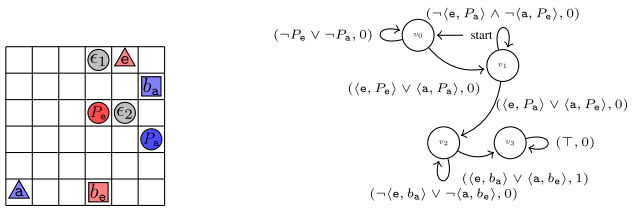



Fig. 3. StochQ-RM uses the LLM-generated DFA $\mathscr{H}$ to learn an equivalent to the ground truth RM.




(a) (b)

Fig. 4. (a) Environment of zero-sum game (b) RM of the general-sum game. $P_e$ and $P_a$ are the powerhouses of agents e and a, respectively. $b_e$ and $b_a$ are the bases. $\epsilon_1$ and $\epsilon_2$ are neutral bases used to make the learning of the RM more challenging.

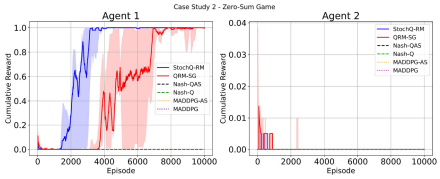



Fig. 5. StochQ-RM converges to an optimal policy faster compared to the baselines by using the LLM-generated DFA.

$(\mathrm{x} = 0, \mathrm{y} = 3)$ in the grid (origin is at the left bottom corner similar to Figure 2(a)). The LLM-generated DFA for agents e and a is $\{\zeta_w, \Upsilon_w\}$ where $w \in \{1, 2\}$, and $\{\beta_w, \Xi_w\}$, with $\Xi_w$ representing the added sub-task. Figure 6 shows StochQ-RM converging to an optimal policy faster by using LLM-generated DFA for both agents, while the baselines fail since the sparsity of the events is higher in this case.

## IX. Conclusion

We showed that our proposed algorithm StochQ-RM outperforms the baselines for both general-sum and zero-sum games.

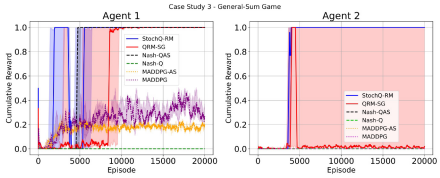



Fig. 6. StochQ-RM converges to an optimal policy faster by using LLM-generated DFA.

Moreover, we showed that the StochQ-RM converges to an optimal policy by using LLM-generated DFA $\mathscr{H}$.

## References

[1] R. T. Icarte, T. Klassen, R. Valenzano, and S. McIlraith, "Using reward machines for high-level task specification and decomposition in reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 2107–2116.

[2] Y. Zhu et al., "Large language models for information retrieval: A survey," 2023, *arXiv:2308.07107*.

[3] J. Wei et al., "Chain-of-thought prompting elicits reasoning in large language models," in *Proc. 36th Conf. Neural Inf. Process. Syst.*, vol. 35, pp. 24824–24837, 2022.

[4] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large language models are zero-shot reasoners," in *Proc. 36th Conf. Neural Inf. Process. Syst.*, vol. 35, 2022, pp. 22199–22213.

[5] T. Brown et al., "Language models are few-shot learners," in *Proc. 36th Conf. Neural Inf. Process. Syst.*, vol. 33, pp. 1877–1901, 2020.

[6] R. Rafailov, A. Sharma, E. Mitchell, S. Ermon, C. D. Manning, and C. Finn, "Direct preference optimization: Your language model is secretly a reward model," 2023, *arXiv:2305.18290*.

[7] R. Azamfirei, S. R. Kudchadkar, and J. Fackler, "Large language models and the perils of their hallucinations," *Crit. Care*, vol. 27, no. 1, p. 120, 2023.

[8] C. E. Lemke and J. T. Howson, "Equilibrium points of bimatrix games," *J. Soc. Ind. Appl. Math.*, vol. 12, no. 2, pp. 413–423, 1964.

[9] J. Hu and M. P. Wellman, "Nash Q-learning for general-sum stochastic games," *J. Mach. Learn. Res.*, vol. 4, pp. 1039–1069, Nov. 2003.

[10] R. Lowe, Y. I. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Proc. 31st Conf. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–12.

[11] J. Hu, J.-R. Gaglione, Y. Wang, Z. Xu, U. Topcu, and Y. Liu, "Reinforcement learning with reward machines in stochastic games," 2023, *arXiv:2305.17372*.

[12] S. M. Alsadat, N. Baharisangari, Y. Paliwal, and Z. Xu, "Distributed reinforcement learning for swarm systems with reward machines," in *Proc. Amer. Control Conf. (ACC)*, 2024, pp. 33–38.

[13] Z. Xu et al., "Joint inference of reward machines and policies for reinforcement learning," in *Proc. Int. Conf. Autom. Plan. Sched.*, vol. 30, 2020, pp. 590–598.

[14] D. Hendrycks et al., "Measuring massive multitask language understanding," 2020, *arXiv:2009.03300*.

[15] Y. Yang, J.-R. Gaglione, C. Neary, and U. Topcu, "Automaton-based representations of task knowledge from generative language models," 2022, *arXiv:2212.01944*.

[16] D. Neider, J.-R. Gaglione, I. Gavran, U. Topcu, B. Wu, and Z. Xu, "Advice-guided reinforcement learning in a non-Markovian environment," in *Proc. AAAI Conf. Artif. Intell.*, 2021, pp. 9073–9080.

[17] Z. Xu, Y. Liang, C. Yu, Y. Wang, and Y. Wu, "Fictitious cross-play: Learning global Nash equilibrium in mixed cooperative-competitive games," 2023, *arXiv:2310.03354*.

[18] S. Vemprala, R. Bonatti, A. Bucker, and A. Kapoor, *ChatGPT for Robotics: Design Principles and Model Abilities, Autonomous Systems and Robotics Research*, vol. 2, Microsoft, Redmond, WA, USA, 2023.