International Journal of Mathematical, Engineering and Management Sciences
Vol. 11, No. 2, 505-524, 2026
https://doi.org/10.33889/IJMEMS.2026.11.2.021

A Q-Learning Based Multi-Agent Mechanism for Collusion Tolerance in Voting-Based Systems

Noah Oikarinen
Department of Electrical and Computer Engineering,
University of Massachusetts, Dartmouth, MA, USA.
E-mail: noikarinen@umassd.edu

Liudong Xing
Department of Electrical and Computer Engineering,
University of Massachusetts, Dartmouth, MA, USA.
Corresponding author: lxing@umassd.edu

(Received on November 13, 2025; Revised on January 6, 2026; Accepted on January 17, 2026)

Abstract
Many critical systems use voting-based mechanisms to mask or tolerate faults and improve system reliability. However, such systems are susceptible to collusion attacks where collaborating malicious resources may produce identical erroneous results, potentially causing the task to fail. To detect and alleviate the consequence of such coordinated adversarial behavior, we develop a multi-agent collusion tolerance system, systematically integrating Q-learning based agent allocation, dual-stage spot checking, and strategic credibility adjustment. A spotter agent is in charge of detecting a malicious resource agent based on its performance at the pre-task execution stage and at the post-voting stage. In the case of collusion being detected, the credibility scores of collusive participants are revoked, excluding them from future allocations and task execution. To balance resource utilization and defense effectiveness, optimally allocating spotter agents and resource agents is a crucial and challenging problem. We address this challenge by using reinforcement learning, particularly, Q-learning to optimally allocate these agents under a predefined cost constraint. Comprehensive experiments are performed to illustrate the effectiveness of the proposed collusion tolerance mechanism. A sensitivity analysis of key model parameters is also conducted to demonstrate their impact on the performance of the proposed mechanism.

Keywords- Collusion tolerance, Credibility, Multi-agent system, Q-learning, Reliability.

1. Introduction

In many critical applications, voting-based mechanisms have been employed to mask or tolerate the occurrence of faults, where outputs from multiple voting units collectively determine the final output based on a certain voting method such as the majority voting, threshold voting, plurality voting, and weighted voting (Avizienis et al., 1971; Nordmann and Pham, 1999; Parhami, 1994). Examples include but are not limited to N-version programming (Chen and Avizienis, 1978), imprecise information handling (Ivanov et al., 2016), signal processing in multi-channel communications (Miao et al., 2024), safety monitoring (Chaisawat and Vorakulpipat, 2020), pattern analysis (Tasci et al., 2021), target recognition (Zhang and Zhou, 2023), node selection in edge computing (Chen et al., 2025), image processing (Iqbal et al., 2025; Pilar, 2025), and consensus protocols in blockchains (Gaur et al., 2025; Lei et al., 2025; Verma et al., 2025).

Although the voting mechanism can effectively tolerate faults and improve system reliability, its implementation depends on an assumption that all the voting units are honest. In practice, multiple voting units may cooperate through information sharing and aligning their votes to produce the same incorrect result, undermining task reliability and system integrity. In the event of enough dishonest voting units colluding, the incorrect result wins the votes, causing the task execution to fail. Such attacks are referred to as collusion attacks, and their impact must be addressed for designing and implementing a robust voting-based system.

For effective design and analysis, each voting unit can be modeled as an agent, and the entire system is modeled as a multi-agent system (MAS). In the general context, multiple autonomous agents in an MAS interact and execute tasks in parallel, offering resilience against individual agent failures (Li et al., 2024; Wang et al., 2024; Zhao et al., 2024). In the context of voting-based MAS, task reliability can be compromised by collusion attacks where multiple malicious agents coordinate to manipulate or corrupt results (Bonjour et al., 2022; Tassa et al., 2019). Being able to detect and mitigate the consequence of collusion attacks plays a pivotal role in ensuring system reliability and integrity (Owoputi and Ray, 2022).

Some efforts have been devoted to addressing collusion attacks in MAS. For instance, a data-driven dynamic contract method was suggested for identifying and disincentivizing collusion agents in Aguiar et al. (2022). The need for mitigating secret collusion among generative artificial intelligence (AI) agents was investigated in Motwani et al. (2024). The need for detecting collusion in iterated games by Q-learning agents was examined in Bertrand et al. (2023). A randomized communication topology-based method was suggested to prevent malicious agents from exploiting predictable communication channels to launch collusion attacks in a blockchain system (Li et al., 2023). A collusion-proof Delegated Proof-of-Stake consensus protocol based on a selection pressure algorithm was proposed in Wang et al. (2023), where a credibility system was adopted to penalize dishonest agents. A spot-checking technique, where a portion of resource agents is randomly chosen for honesty verification, was presented in Staab and Engel (2009) and implemented for mitigating collusion attacks in a grid computing environment (Levitin et al., 2018, 2019), peer-to-peer networks (Wang et al., 2018), and crowd sourcing (Wang et al., 2020). The agents that do not pass the check are isolated or removed from task executions. However, these spot checking-based methods have a limitation that collusive agents that evade detection can still manipulate and corrupt the outcome. Recent work in Oikarinen and Xing (2025) proposed a dual-stage spot-checking approach that verifies a portion of prospective voting agents pre-task execution and conducts a second check of a randomly selected majority voting agent after the task execution and voting, thereby lowering the probability of undetected collusive agents impacting the final task outcome. Despite these efforts, significant challenges persist in developing adaptive, cost-efficient frameworks with intelligent resource allocation, and strategic and iterative credibility adjustment in response to varying attack intensities and detection statuses.

This work aims to address these challenges by developing a multi-agent defense system that integrates reinforcement learning (RL), particularly Q-learning with dual-stage spot checking and strategic credibility adjustment for effectively detecting and mitigating collusion attacks. RL is a subset of machine learning where an agent learns a policy to maximize the reward function in an environment (Sutton and Barto, 2018). The agent takes an action, receives a reward or penalty for that action and updates its behavior depending on the reward given to the action. Over many simulations, the agent improves decision making to maximize the reward. Among RL methods, Q-learning is widely studied, effective for learning problems, and straightforward to implement and analyze (Watkins and Dayan, 1992), making it a good choice to use for diverse problems. For example, prior work has applied Q-learning for dynamic channel assignment in cellular networks (Nie and Haykin, 1999), interference avoidance in self-organized femtocell networks (Bennis and Niyato, 2010) and energy-efficient task scheduling in cloud computing (Ding et al., 2020). Additionally, recent work combines Q-learning with an additional optimizer like particle swarm optimization (PSO), genetic algorithms and FOX optimization to enhance convergence speed and solution performance. In these hybrid methods, the optimizer searches over policy structures, hyperparameters, or allocation combinations, while Q-learning updates action values from interaction. For example, Pang et al. (2025) integrated PSO with Q-learning for electric vehicle charging, using Q-learning to dynamically adjust inertia and learning factors used to enhance Q-learning convergence and scheduling performance. Yang et al. (2025) combined Q-learning with a multi-objective genetic algorithm, which adapts parameters online and achieves a faster convergence and better solutions in flexible job scheduling. Jumaah et al. (2025) used

Q-learning FOX (Q-FOX) optimization (Mohammed and Rashid, 2023) to tune Q-learning hyperparameters online, adaptively selecting the learning rate and discount factor using a fitness objective over average Q-values and outperforming fixed settings in cumulative reward. Our research does not use any additional optimizer because Q-learning is efficient enough and converges to an acceptable outcome quickly.

Prior work in Oikarinen and Xing (2025) combined a dual-stage spot checking with a rule-driven systemic allocation of spotter and resource agents under a total cost constraint. Compared to single-stage spot-checking methods, this design improved collusion detection performance. However, the allocation adapted slowly when collusion was heavy and it was designed to spend the entire cost constraint even after most colluders had been removed. The system also would achieve task failure in early episodes when it is needed to allocate more spotter agents and it would use the entire cost constraint even when the pool had no colluding agents in it. The proposed research is needed to add the missing adaptivity. We employ Q-learning to learn a policy that is cost aware and changes allocation in response to the current state of the system. More specifically, the master agent learns to allocate more spotter agents in early episodes when there is a high amount of collusion and then scale back on the number of agents allocated when the credibility scores are high and the number of colluding agents is low, which raises task success rate without using the entire cost constraint. Training is done across a large number of simulations so the master agent experiences diverse collusion patterns and learns how to defend against them in different ways (using different allocation strategies). To verify the performance of the proposed Q-learning-based collusion detection MAS, comprehensive experiments are conducted. A sensitivity analysis of several key parameters is also performed, demonstrating their impact on the performance of the proposed defense system. Note that in the proposed model, collusive agents do not adapt or behave strategically beyond coordinating to align their votes and submit the same incorrect vote.

The rest of the paper is organized as follows. Section 2 presents the proposed collusion defense method. Section 3 studies its performance under varying parameters including honest agent's reliability, costs of spotter and resource agents, level of collusion (or the number of collusive agents), reward/penalty combinations, total cost requirement/constraint, initial credibility score value, and efficiency bonus. Section 4 summarizes the findings and outlines future research problems.

2. Proposed Defense MAS

This section presents the proposed collusion defense MAS, integrating Q-learning-based agent allocation under a pre-specified cost constraint (C^*), strategic credibility score updating, and dual-stage spot-checking mechanisms. The MAS consists of a master agent in charge of the agent allocation, i.e., determining the number of spotter agents (N_s) and the number of resource agents (N_r) based on a pool of heterogeneous agents. A spotter agent is responsible for verifying the honesty of a resource agent (i.e., identifying collusive resource agents). Each resource agent, if identified as a work agent, is responsible for executing the real task and supplies its result to the voting agent, which makes decision on the final output. **Figure 1** summarizes the different types of agents in the proposed MAS.

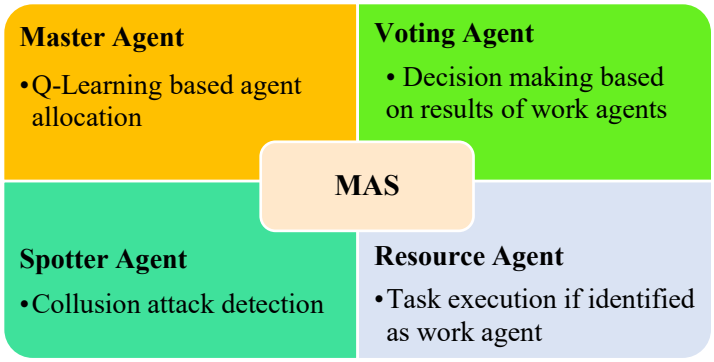

Figure 1. Types of agents and their roles.

2.1 Q-Learning-based Agent Allocation

The master agent uses Q-learning to find an optimal sequence of spotter and resource agent allocations under different levels of collusion. During training, the master agent runs multiple simulations, each made up of multiple episodes. In each episode, the master agent observes the current state, selects an action (N_s, N_r) that satisfies the cost constraint, and receives the task success metric TSM (defined in Section 2.3) as the reward. At the end of each simulation, the master agent computes the reliability R defined in Equation (1) and uses R as a quality assurance for the simulation.

$$R = \frac{\text{Total Number of Successful Episodes}}{\text{Total Number of Episodes}} \tag{1}$$

If the calculated R meets or is over the quality threshold, the corresponding transitions for that simulation are used to update the Q-table and the policy. Over multiple training simulations, the master agent converges towards a set of spotter and resource agent allocation pairs.

The Q-learning model used in this work is formulated in Equation (2), where $Q(S, A)$ is current Q-value for state $S = (TSM, N_s, N_r)$ and action $A = (N_s, N_r)$, α is the learning rate, TSM is the reward obtained after taking an action, γ is the discount factor, and $maxQ(S', A')$ is the maximum Q-value predicted for the next state S'.

$$Q(S, A) \leftarrow Q(S, A) + \alpha\big(TSM + \gamma\, maxQ(S', A') - Q(S, A)\big) \tag{2}$$

More specifically, at each episode t in the simulation, the master agent tracks the previous episode's TSM, N_s and N_r so the master agent can learn to tell the severity of the collusion in relation to the episode. The action space is a set of valid (N_s, N_r) pairs generated to satisfy the cost constraint C^*, as formulated in Equation (3).

$$N_s * C_s + N_r * C_r \leq C^* \tag{3}$$

where, C_s and C_r are the cost of a spotter agent and a resource agent, respectively, $N_s \geq 1$, $N_r \geq 3$ (to make the majority voting work), and $N_r \geq N_s$.

During training, the master agent follows an ε-greedy policy. With probability ε, it explores by choosing random (N_s, N_r) allocation, and with probability $(1 - \varepsilon)$, it exploits by selecting the action with the highest Q value for the state space. At the start, ε is large and is decayed over episodes so the policy will converge. This greedy policy balances trying new actions with using best-known actions and helps avoid getting stuck in a local maximum policy.

The memory cost of tabular Q-learning grows with the size of the state and action spaces. It stores a Q-table with $|S| \times |A|$ entries, giving a space complexity of $O(|S||A|)$. In each episode during action selection, the master agent scans the feasible actions and chooses the one with the highest Q-value, giving $O(|A|)$ time, and the Q-value update is $O(1)$.

2.2 Strategic Credibility Score Adjustment and Spot Checking

Each resource agent i in the pool is initialized with credibility score S_i, which is subject to updating based on detection of collusion behavior during spot checking and task outcomes. Following the agent allocation, N_r resource agents are selected from the agent pool based on their credibility scores. Particularly, any agent i can be selected with probability (P_i) proportional to its credibility score (S_i), as defined in Equation (4).

$$P_i = \frac{S_i}{\sum_{l=1}^{K} S_l} \tag{4}$$

where, K is the number of agents in the pool with a positive credibility score ($S_l > 0$). This probabilistic selection policy ensures that more credible resource agents are more likely to participate in task execution.

This work assumes that all spotter agents are perfect, thus they have no credibility scores. In practical systems, spot checking may be imperfect and can have false negative and false positive errors. A false negative error occurs when a collusive agent passes a spot checking and remains in the system, which can allow collusive agents to continue participating in voting after the first spot checking and can also cause an incorrect output to be accepted after the second spot checking. A false positive takes place when an honest agent is incorrectly flagged as colluding and penalized, which can remove the honest agent from voting after the first spot checking or cause a correct output to be falsely deemed as incorrect after the second spot checking. These imperfections would generally reduce both TSM and R. Both types of errors will be considered in our future work. Following the selection of N_r resource agents from the pool, the first spot-checking stage begins, where each spotter agent assesses one randomly picked resource agent, for example, through assigning a task with a known correct answer. If the answer provided by the checked resource agent does not match the desired correct answer, then this resource agent is identified as collusive and its credibility score is reset to zero, making it be excluded from future allocations and thus task execution. A resource agent becomes a work agent to execute the real task if it is selected for spot checking and passes the test, or not picked for the check due to the shortage of spotter agents (from the allocation constraint $N_r \geq N_s$).

After the first spot-checking stage, the work agents execute the real task and supply their answers or votes on task outcomes to the voting agent, which determines the task output based on certain voting mechanism. In this work, a majority voting mechanism is implemented, which requires at least three votes from working agents. When a tie occurs, no output is decided from the voting and no credibility score update is performed. In this case, the task fails, and the episode is considered unsuccessful. If there are fewer than three work agents after the first spot checking, all agents are returned to the agent pool; the episode is reset and is considered an unsuccessful run.

In the case of an output being successfully produced by the voting agent, a second spot-checking stage starts, aimed at alleviating the risk of a malicious output occurring (i.e., the majority of work agents executing the task being collusive agents). During this stage, the spotter agent randomly chooses a winning work agent (i.e., whose answer matches the voted output) for collusion check. There are two outcomes:

- If the checked agent is found to be collusive, the task is failed and all winning work agents' credibility scores are set to 0, making them be excluded from future agent allocations.

- If the checked agent passes the check, the task is successful and all participating work agents' credibility scores are adjusted. Specifically, as formulated in Equation (5), the credibility score of each winning agent (whose answer matches the voted output) is increased by a constant value (ΔS) while the credibility score of each agent whose answer does not match the voted output) is decreased by a constant value ($\Delta S'$).

$$S_i^{new} = \begin{cases} S_i^{old} + \Delta S & \text{if } i \in \text{Work agents in the majority vote} \\ S_i^{old} + \Delta S' & \text{if } i \in \text{Work agents in the minority vote} \end{cases} \quad (5)$$

The above credibility score adjustment policies reward agents that make positive contributions to task success, penalize agents that produce wrong results, and exclude malicious agents systematically from impacting future task executions.

2.3 TSM Definition

Depending on the task execution status, an award in the form of a task success metric (TSM) is defined as formulated in Equation (6).

$$TSM_{raw} = \begin{cases} \frac{N_a}{N_v} + \lambda * \left(1 - \frac{N_s * C_s + N_r * C_r}{C^*}\right) & \text{if task succeeds} \\ -\frac{N_a}{N_v} + \lambda * \left(1 - \frac{N_s * C_s + N_r * C_r}{C^*}\right) & \text{if task fails} \end{cases} \quad (6)$$

where, N_a is the number of winning work agents, N_v is the total number of work agents or votes, and λ is an efficiency bonus parameter rewarding the master agent for using fewer agents (less total cost). Expression (7) is used to normalize TSM_{raw} to a score in the range [0, 1] using a min-max normalization method (Singh and Singh, 2022), where $TSM_{min} = -1$ and $TSM_{max} = 1 + \lambda$ since TSM_{raw} has a fixed range at $[-1, 1 + \lambda]$.

$$TSM = \frac{TSM_{raw} - TSM_{min}}{TSM_{max} - TSM_{min}} = \frac{TSM_{raw} - (-1)}{(1+\lambda) - (-1)} = \frac{TSM_{raw} + 1}{2 + \lambda} \quad (7)$$

2.4 Systematic Defense Mechanism

Based on the discussions in preceding subsections, we summarize the collusion defense procedure within each episode in **Figure 2** and describe it as a step-by-step procedure as follows.

1) The user supplies cost constraint (C^*) to the master agent.
2) Based on Q-table, the master agent determines the allocation (N_s, N_r) meeting C^*.
3) N_s spotter agents are selected from the agent pool.
4) N_r resource agents are probabilistically selected from the agent pool, with probability defined by Equation (4).
5) Each spotter agent conducts the first check on a randomly selected resource agent.
6) Resource agents that are chosen for collusion check and pass the check are promoted to work agents. Resource agents that are not chosen during the spot checking are also promoted to work agents. Those that are chosen but fail the check are identified as collusive and are excluded from task execution by setting their credibility scores to zero.
7) If there are 3 or more work agents resulting from the first spot checking, these work agents execute the real task and produce their task outputs; otherwise (less than 3 work agents), they are returned to the agent pool without any credibility score adjustment.
8) The voting agent determines the final output based on votes (task outputs) from work agents.
9) The second spot checking is performed, where a randomly selected winning agent is checked to validate the final task output. In the case of failing the check, the task is failed and the credibility scores of all

winning work agents participating in the majority voting are set to 0; in the case of passing the check, the task is successful and the credibility scores of all winning work agents are incremented by ΔS while the credibility scores of all work agents with output not matching the voted output are decreased by $\Delta S'$.

10) All participating work agents with updated credibility scores return to the agent pool for potential reallocation in future episodes.
11) Update the Q-table using Equation (2).

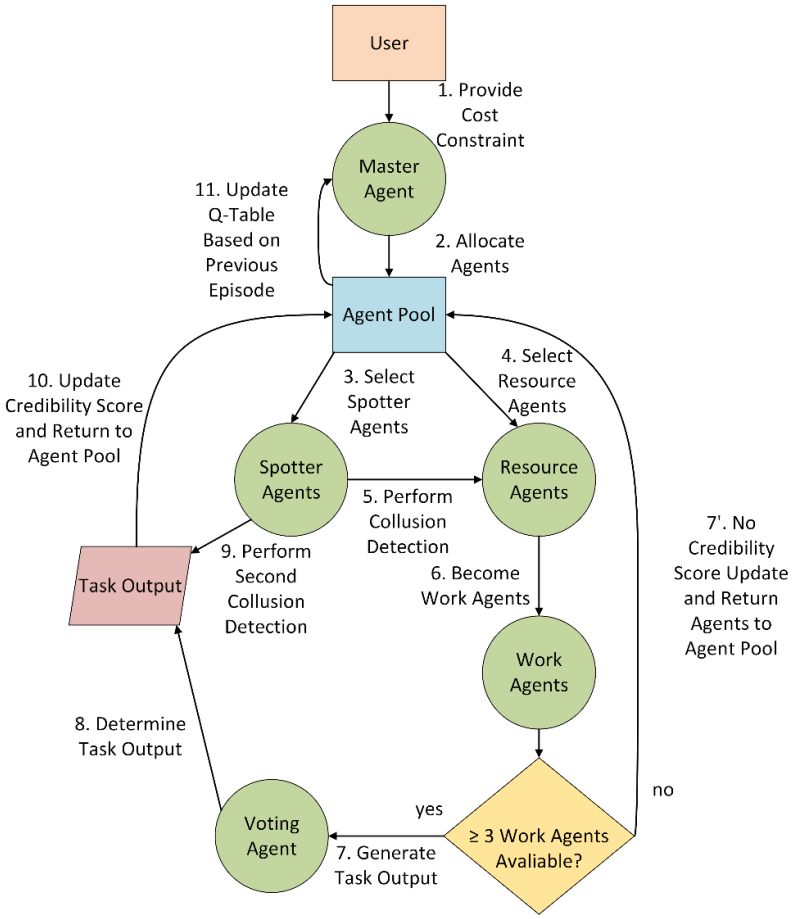

Figure 2. Flow chart of the proposed Q-learning-based defense.

The above procedure is implemented iteratively to maximize the TSM. Instead of using a fixed rule that decreases the number of spotter agents after a success episode in Oikarinen and Xing (2025), the master agent uses a learned policy informed by feedback to adaptively determine the next episode's spotter and resource agent allocation, enabling a continuous and more effective refinement of the defense in evolving multi-agent environments.

In real-world deployment, the scalability of the proposed defense mechanism depends primarily on the task executed by the work agents and on the number of agents allocated per episode. The master agent is lightweight, while the dominant computation and communication costs arise from task execution by work

agents and from performing spot-checking tasks. As the allocated agent counts increase, computation and communication costs generally scale with the number of participating agents. However, communication costs are mitigated because the learned policy is cost-aware and seeks to achieve task success using the minimum necessary agent allocations. The proposed defense mechanism is well-suited for systems that require fast and accurate decisions.

3. Validation and Sensitivity Analysis

We validate the proposed collusion defense MAS and perform sensitivity analysis of key model parameters using comprehensive experiments that involve executing an object detection task. The performance of the proposed mechanism is evaluated using R defined in Equation (1) and TSM defined in Equation (6). R measures the consistency of the proposed defense mechanism in achieving a successful outcome across many episodes while TSM measures the capability of correctly executing the task. For each parameter setting, the master agent is first trained using 1000 training simulations, each consisting of 250 episodes. Convergence is assessed by observing the learned Q-values and selected allocations consistently converge over training simulations. After training, we run 1000 test simulations, each also consisting of 250 episodes, and we compute R and TSM for each test simulation. The values of R and TSM are evaluated as the average over those 1000 test runs. **Table 1** summarizes the default configuration of key parameters used in the simulations. The values in **Table 1** serve as a reference baseline for comparison. All cost values represent relative resource usage only, not real-world cost. In Sections 3.2-3.8, a sensitivity analysis is performed by sweeping parameter values to show how each parameter affects system performance.

Table 1. Default values of model parameters.

Parameter	Value
Reliability of an honest work agent	99.99%
Total cost constraint (C^*)	20
Cost of a resource agent (C_r) *versus* Cost of a spotter agent (C_s)	2:1
Number of collusive agents *versus* Total number of resource agents	50:150
Initial credibility score of each resource agent	5
Reward ΔS	3
Penalty $\Delta S'$	2
Efficiency bonus (λ)	0.3

3.1 Performance Comparison

Under the parameter setup in **Table 1**, TSM and R are obtained as 0.8890 and 0.9348, respectively. For comparisons, we also obtain TSM and R under the defense method in Oikarinen and Xing (2025), which are 0.8571 and 0.9962, respectively. We compare the proposed mechanism against the most recent and directly comparable baseline developed by Oikarinen and Xing (2025), which has been shown to outperform prior methods using single-stage spot-checking. Using this baseline provides a strong reference point and allows an informative comparison. The previous method obtains a higher R because it is not cost aware and uses the entire cost constraint, often allocating more agents than needed to achieve a successful episode. If we similarly ignore cost and tune for reliability, we can achieve similar results for R, but this comes at expense of a lower TSM and reduced cost efficiency. See Section 3.8 on efficiency bonus for additional studies of this trade off. In this work, we prioritize TSM and cost efficiency over R.

In the following subsections, we perform sensitivity analysis to demonstrate the impact of different model parameters on the performance, including the reliability of honest agents, costs of spotter and resource agents, level of collusion (or the number of collusive agents), reward/penalty in credibility score adjustment, total cost requirement/constraint, initial credibility score value, and efficiency bonus.

3.2 Reliability of Honest Agents

Figure 3 and **Table 2** show the values of TSM and R as the honest agent reliability (HAR) changes from 0.05 to 0.9999 in the step of 0.05. As HAR reflects the likelihood of supplying a correct vote (i.e., a correct task execution), it is intuitive that TSM monotonically increases as HAR increases. More specifically, TSM starts off low, but climbs progressively and exceeds 0.8 when HAR reaches 85%. On the other hand, R decreases from about 0.41 at HAR = 5% to about 0.32 at HAR = 25%. After that, R climbs steadily to about 0.53 at HAR = 60%, then following a large jump to about 0.73 at HAR = 65%, it returns to a steady climb that goes up to about 0.94 at HAR = 99.99%, reflecting the system's growing capability of aligning the majority vote with the correct result. When HAR is low (below 60%), R has significantly low values, implying that collusive or unreliable work agents frequently sway votes to incorrect outputs. The initial decrease in R can be explained by the fact that, at very low HAR levels (5~10%), honest agents vote incorrectly most of times and their votes are scattered, often leading to ties that cause R to stay low. As HAR increases to the 15 ~ 25% range, honest agents become more consistent, but the majority of their votes are still wrong. At this point, collusive agents can more easily secure majority vote because honest agents are now more consistent in producing wrong output. This results in more decisive, but incorrect majorities instead of ties, causing R to dip even further than in the HAR being in the 5 ~ 10% range, reaching the lowest value at HAR = 25% with R = 0.3214. After HAR = 25%, honest agents start to function well enough that the correct vote begins to win the majority more often, causing R to steadily rise. Between HAR = 60% ~ 65% where R = 0.5349 to 0.7291, a significant increase occurs as honest agents become accurate enough to consistently win the majority vote and maintain a high credibility score, allowing them to be selected from the pool in later episodes.

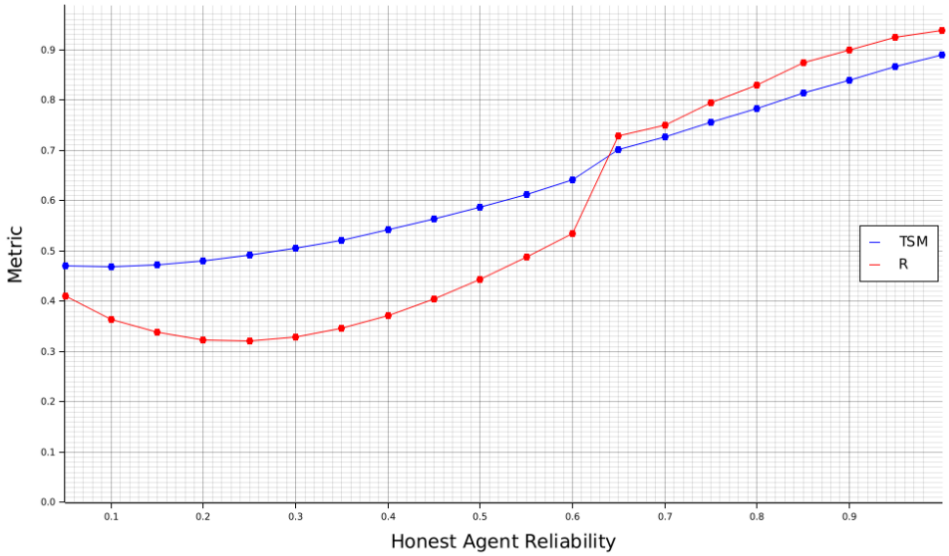

Figure 3. Impact of HAR on system performance metrics.

The trend shows that TSM and R are highly sensitive to HAR. Incorporating more highly reliable honest agents can significantly improve voting outcomes and enhance the system's resilience to collusion attacks.

Table 2. TSM and R under different values of HAR.

HAR	*TSM*	*R*	HAR	*TSM*	*R*
5.00%	0.4696	0.4097	55.00%	0.6127	0.4873
10.00%	0.4689	0.3639	60.00%	0.6404	0.5349
15.00%	0.4727	0.3383	65.00%	0.7007	0.7291
20.00%	0.4802	0.3229	70.00%	0.7266	0.7495
25.00%	0.4912	0.3214	75.00%	0.7554	0.7935
30.00%	0.5049	0.3279	80.00%	0.7831	0.8296
35.00%	0.5215	0.3456	85.00%	0.8135	0.8734
40.00%	0.5413	0.3710	90.00%	0.8397	0.8984
45.00%	0.5631	0.4044	95.00%	0.8666	0.9249
50.00%	0.5868	0.4427	99.99%	0.8892	0.9374

3.3 Costs of Spotter and Resource Agents

Table 3 and **Figure 4** show the values of TSM and R under different cost combinations in the form of C_s : C_r. The system offers high performance levels for combinations with low costs, like 1:1 or 2:1 or 1:2. Under those combinations, the master agent can allocate enough spotter agents to effectively identify collusive agents while assigning sufficient resource agents to execute the real task and tolerate malicious outputs from any remaining collusive agents.

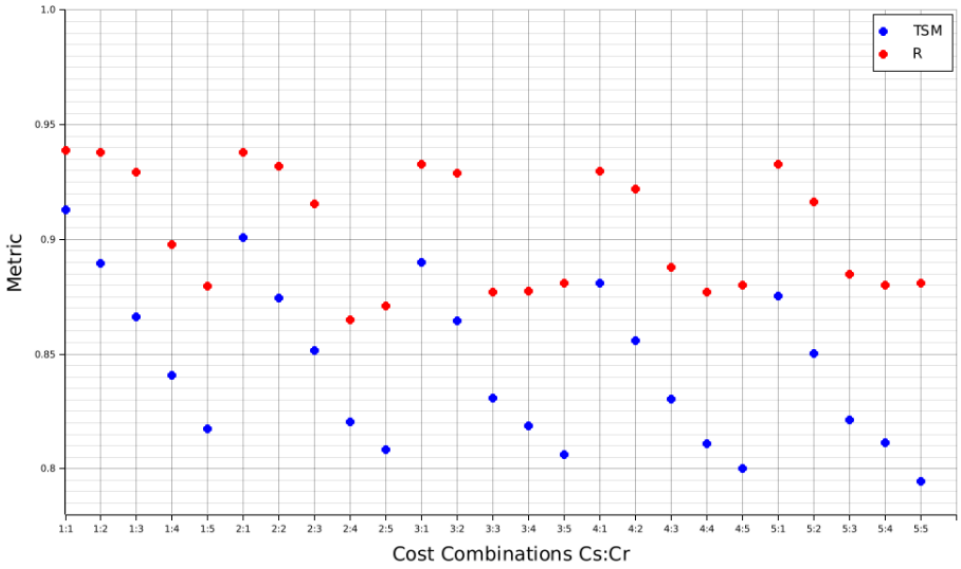

Figure 4. TSM and R under different combinations of spotter and resource agent costs.

At a fixed cost of spotter agent (C_s), increasing the cost of resource agents (C_r) produces a clear decrease in TSM. While the effect of a higher cost of resource agents ($C_r > 3$) on R varies with different fixed C_s values due to the factor that how many work agents can be supported within the overall cost constraint C^*. For $C_s = 2$ to 5, there is a consistent small uptick in R from $C_r = 4$ to 5 and the size of this uptick shrinks as C_s increases. This uptick may come from a change in policy and discrete allocation effects in the system. At C_s = 2, 3, 4, the policy often shifts from using more spotter agents at C_r = 4 to using fewer spotter agents at C_r = 5. Fewer spotter agents reduce the risk of task failures at early episodes; particularly, the likelihood of ties or having fewer than three work agents after the first spot checking (leading to task failures) can be

reduced, for example, when four resource agents are allocated together with one spotter agent instead of two spotter agents. So, a slightly larger amount of successful episodes happens and R rises even though TSM falls because the cost efficiency is lower. At $C_s = 5$, both $C_r = 4$ and 5 force the same amount of spotter and resource agent allocation, causing the master agent to fall in a pattern of minimal spotter allocation to satisfy the minimum agent allocation requirement. So, the small uptick can be due to the efficiency bonus not being enough to make up for the one cost difference and the amount of successful episodes being more when $C_r = 5$. The $C_r = 4$ to $C_r = 5$ uptick can also be due to the fact that when $C_r = 5$, the master agent allocates fewer agents; as a result, after the first spot checking, there is a higher chance of removing a work agent falling below the minimum of three work agents, causing the TSM not to be calculated. Overall, when C_s is expensive, there are fewer flexible allocations under heavy collusion, causing the reliability to be significantly lower.

Table 3. TSM and R under different cost combinations.

$C_s : C_r$	TSM	R	$C_s : C_r$	TSM	R
1 : 1	0.9131	0.9387	4 : 1	0.8811	0.9299
1 : 2	0.8896	0.9380	4 : 2	0.8559	0.9221
1 : 3	0.8662	0.9294	4 : 3	0.8303	0.8880
1 : 4	0.8408	0.8976	4 : 4	0.8110	0.8772
1 : 5	0.8176	0.8796	4 : 5	0.7999	0.8801
2 : 1	0.9009	0.9379	5 : 1	0.8753	0.9330
2 : 2	0.8744	0.9318	5 : 2	0.8502	0.9163
2 : 3	0.8514	0.9154	5 : 3	0.8211	0.8846
2 : 4	0.8205	0.8648	5 : 4	0.8115	0.8800
2 : 5	0.8085	0.8711	5 : 5	0.7945	0.8811
3 : 1	0.8901	0.9327			
3 : 2	0.8644	0.9291			
3 : 3	0.8308	0.8772			
3 : 4	0.8188	0.8774			
3 : 5	0.8060	0.8810			

3.4 Collusion Level

The agent pool initially has 150 resource agents available for agent allocations. To examine the impact of collusion level on the system performance, we evaluate TSM and R by varying the number of collusive agents (denoted by N_c) from 1 to 149, as summarized in **Table 4** and demonstrated in **Figure 5**. It is intuitive that when the collusion level is relatively low, both TSM and R remain high. As the number of collusive agents increases, both performance metrics decrease. A steep fall occurs when the number of collusive agents exceeds 120 and a sharp decline at 148 collusive agents. The two curves cross at 93 collusive agents. The reason for the two metrics crossing is because TSM is normalized to be in the range of 0 to 1 with a tie being about 0.5 while R is different (it naturally is defined to be between 0 and 1). If TSM was not normalized, both TSM and R will have similar magnitudes. TSM and R have the same overall shape, but the normalization rescales TSM so it follows the same trend but with a different magnitude. Both TSM and R still remain high even with a high level of collusion due to the rapid elimination of collusive agents in early episodes. Specifically, in these early episodes, the master agent learns a policy that allocates more spotter agents to accelerate the removal of collusive agents, then in later episodes uses fewer agents so a lower number of honest agents can still succeed. When the number of collusive agents is 148 or 149 (under heavy collusion), the system cannot operate properly and honest agents cannot successfully vote for a successful task.

Table 4. TSM and R under different numbers of collusive agents.

N_c	TSM	R	N_c	TSM	R	N_c	TSM	R
1	0.9157	0.9995	51	0.8875	0.9327	101	0.8208	0.8130
2	0.9154	0.9990	52	0.8867	0.9326	102	0.8200	0.8129
3	0.9155	0.9984	53	0.8857	0.9317	103	0.8167	0.8052
4	0.9143	0.9979	54	0.8841	0.9254	104	0.8168	0.8049
5	0.9153	0.9971	55	0.8837	0.9290	105	0.8121	0.7963
6	0.9156	0.9961	56	0.8838	0.9280	106	0.8143	0.8043
7	0.9161	0.9957	57	0.8802	0.9199	107	0.8126	0.7981
8	0.9142	0.9957	58	0.8806	0.9210	108	0.8074	0.7870
9	0.9153	0.9939	59	0.8777	0.9142	109	0.8075	0.7891
10	0.9151	0.9937	60	0.8785	0.9192	110	0.8062	0.7846
11	0.9157	0.9922	61	0.8773	0.9160	111	0.8019	0.7754
12	0.9137	0.9914	62	0.8741	0.9095	112	0.8014	0.7792
13	0.9142	0.9907	63	0.8721	0.9051	113	0.7968	0.7709
14	0.9140	0.9897	64	0.8732	0.9079	114	0.7996	0.7760
15	0.9133	0.9893	65	0.8718	0.9065	115	0.7970	0.7691
16	0.9138	0.9864	66	0.8720	0.9070	116	0.7959	0.7676
17	0.9122	0.9878	67	0.8688	0.9011	117	0.7935	0.7635
18	0.9126	0.9857	68	0.8676	0.8975	118	0.7878	0.7522
19	0.9125	0.9852	69	0.8669	0.8968	119	0.7851	0.7467
20	0.9124	0.9851	70	0.8637	0.8907	120	0.7827	0.7460
21	0.9115	0.9843	71	0.8637	0.8920	121	0.7743	0.7303
22	0.9113	0.9815	72	0.8618	0.8861	122	0.7770	0.7371
23	0.9110	0.9800	73	0.8606	0.8854	123	0.7671	0.7210
24	0.9094	0.9778	74	0.8584	0.8785	124	0.7637	0.7222
25	0.9093	0.9792	75	0.8573	0.8797	125	0.7520	0.7006
26	0.9089	0.9762	76	0.8583	0.8806	126	0.7517	0.7036
27	0.9087	0.9748	77	0.8541	0.8746	127	0.7379	0.6818
28	0.9078	0.9756	78	0.8536	0.8709	128	0.7293	0.6669
29	0.9068	0.9740	79	0.8526	0.8719	129	0.7256	0.6604
30	0.9067	0.9713	80	0.8515	0.8683	130	0.7219	0.6610
31	0.9061	0.9738	81	0.8495	0.8632	131	0.7192	0.6487
32	0.9052	0.9705	82	0.8494	0.8637	132	0.7147	0.6392
33	0.9043	0.9670	83	0.8478	0.8602	133	0.7111	0.6323
34	0.9037	0.9661	84	0.8459	0.8591	134	0.7000	0.6100
35	0.9030	0.9641	85	0.8462	0.8604	135	0.6895	0.5838
36	0.9033	0.9656	86	0.8418	0.8478	136	0.6877	0.5675
37	0.9018	0.9627	87	0.8404	0.8488	137	0.6736	0.5393
38	0.9010	0.9611	88	0.8403	0.8496	138	0.6753	0.5365
39	0.8998	0.9580	89	0.8380	0.8427	139	0.6738	0.5319
40	0.8995	0.9572	90	0.8387	0.8453	140	0.6658	0.5252
41	0.8987	0.9557	91	0.8366	0.8429	141	0.6763	0.5410
42	0.8976	0.9530	92	0.8354	0.8402	142	0.6872	0.5692
43	0.8966	0.9522	93	0.8326	0.8326	143	0.6887	0.5619
44	0.8947	0.9486	94	0.8316	0.8333	144	0.6947	0.5741
45	0.8954	0.9489	95	0.8284	0.8265	145	0.6902	0.5426
46	0.8939	0.9461	96	0.8276	0.8236	146	0.6715	0.4957
47	0.8917	0.9414	97	0.8258	0.8209	147	0.5981	0.4050
48	0.8918	0.9429	98	0.8252	0.8201	148	0.4356	0.0051
49	0.8905	0.9390	99	0.8224	0.8138	149	0.4342	0.0000
50	0.8878	0.9328	100	0.8225	0.8141			

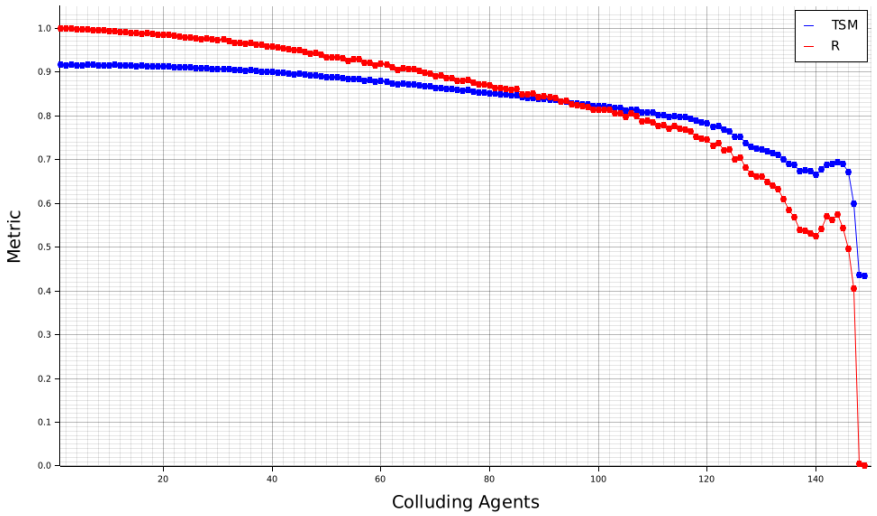

Figure 5. TSM and R under different numbers of collusive agents.

3.5 Credibility Score Reward and Penalty

Figure 6 and **Table 5** demonstrate TSM and R under varying credibility score reward and penalty combinations. For a fixed ΔS, increasing the penalty $\Delta S'$ yields consistent gains in both TSM and R. Similarly, keeping a fixed $\Delta S'$ and increasing the reward ΔS also show increases in both TSM and R.The fluctuations in TSM range from about 0.88 to 0.9, and around 0.92 to 0.95 for R. A larger penalty like $\Delta S' = 5$ does get rid of collusive agents more effectively, enhancing the overall performance but with the risk of over penalizing honest agents who have voted incorrectly. A larger reward like $\Delta S = 5$ properly rewards the honest agents more effectively, yielding a better increase in the chances of being picked again for future tasks.

Table 5. TSM and R under different reward and penalty combinations.

$+\Delta S/-\Delta S'$	TSM	R	$+\Delta S/-\Delta S'$	TSM	R
+1/-1	0.8807	0.9248	+4/-1	0.8889	0.9364
+1/-2	0.8849	0.9309	+4/-2	0.8908	0.9378
+1/-3	0.8860	0.9291	+4/-3	0.8926	0.9413
+1/-4	0.8879	0.9323	+4/-4	0.8946	0.9447
+1/-5	0.8934	0.9379	+4/-5	0.8967	0.9460
+2/-1	0.8839	0.9311	+5/-1	0.8919	0.9439
+2/-2	0.8856	0.9301	+5/-2	0.8940	0.9456
+2/-3	0.8886	0.9355	+5/-3	0.8938	0.9445
+2/-4	0.8900	0.9355	+5/-4	0.8954	0.9449
+2/-5	0.8928	0.9368	+5/-5	0.8974	0.9461
+3/-1	0.8865	0.9326			
+3/-2	0.8893	0.9388			
+3/-3	0.8913	0.9396			
+3/-4	0.8927	0.9412			
+3/-5	0.8952	0.9428			

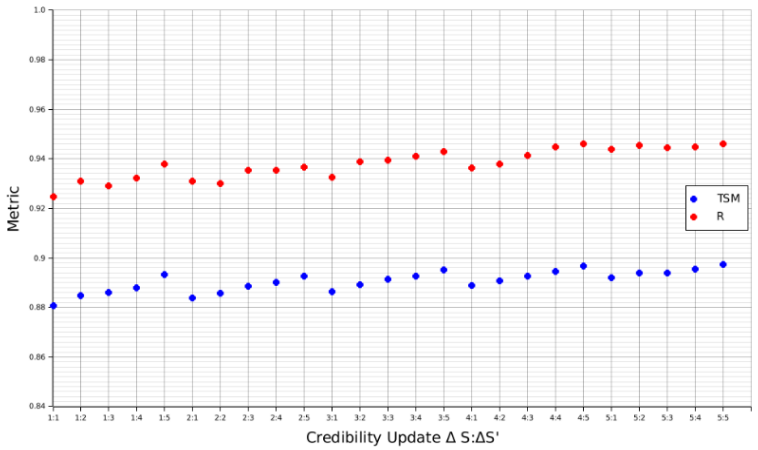

Figure 6. TSM and R under different reward and penalty combinations.

3.6 Total Cost Constraint

The total cost constraint dictates the resource available for assigning spotter agents and resource agents during each episode. It varies from 7 to 50 in this case study. **Figure 7** and **Table 6** shows that increasing C^* raises both TSM and R in general.

Table 6. TSM and R under different total cost constraints.

C^*	TSM	R	C^*	TSM	R
7	0.7943	0.8807	29	0.9080	0.9385
8	0.8046	0.8708	30	0.9087	0.9356
9	0.8089	0.8600	31	0.9114	0.9383
10	0.8190	0.8601	32	0.9133	0.9400
11	0.8338	0.8859	33	0.9141	0.9410
12	0.8479	0.9141	34	0.9155	0.9401
13	0.8562	0.9199	35	0.9152	0.9359
14	0.8640	0.9303	36	0.9172	0.9381
15	0.8698	0.9336	37	0.9184	0.9397
16	0.8741	0.9313	38	0.9190	0.9382
17	0.8781	0.9327	39	0.9207	0.9407
18	0.8819	0.9340	40	0.9218	0.9415
19	0.8854	0.9318	41	0.9221	0.9411
20	0.8893	0.9385	42	0.9247	0.9455
21	0.8919	0.9382	43	0.9255	0.9451
22	0.8939	0.9347	44	0.9274	0.9485
23	0.8967	0.9362	45	0.9292	0.9512
24	0.8991	0.9360	46	0.9285	0.9484
25	0.9004	0.9349	47	0.9301	0.9500
26	0.9029	0.9370	48	0.9303	0.9502
27	0.9056	0.9391	49	0.9332	0.9560
28	0.9070	0.9383	50	0.9327	0.9533

Specifically, at $C^* = 7$ and 8, TSM decreases because the budget can support two spotter agents, which raises the chance that the first spot checking detects and removes a collusive agent (by setting its credibility score to 0), and if fewer than three work agents remain after the check, the episode fails and no TSM is calculated, but this episode is still counted towards evaluating R as an unsuccessful episode. From $C^* = 9$ to 15, TSM climbs from about 0.81 to about 0.87 and R increases from 0.86 to about 0.93 as the master

agent is forced to choose allocations of fewer spotter and resource agents, leaving the system vulnerable to collusion due to the cost constraint being so low. Between $C^* = 16$ and 30, both TSM and R continue to improve, with R growing gradually reaching about 0.93 and TSM growing more quickly reaching about 0.91. Beyond $C^* = 30$, the gains start to taper with both TSM and R continuing a gradual climb reaching about 0.93 and 0.95 at $C^* = 50$, respectively. Overall, a larger C^* allows the master agent to allocate more spotter and resource agents in the early episodes when the system is under heavy collusion, improving both TSM and R.

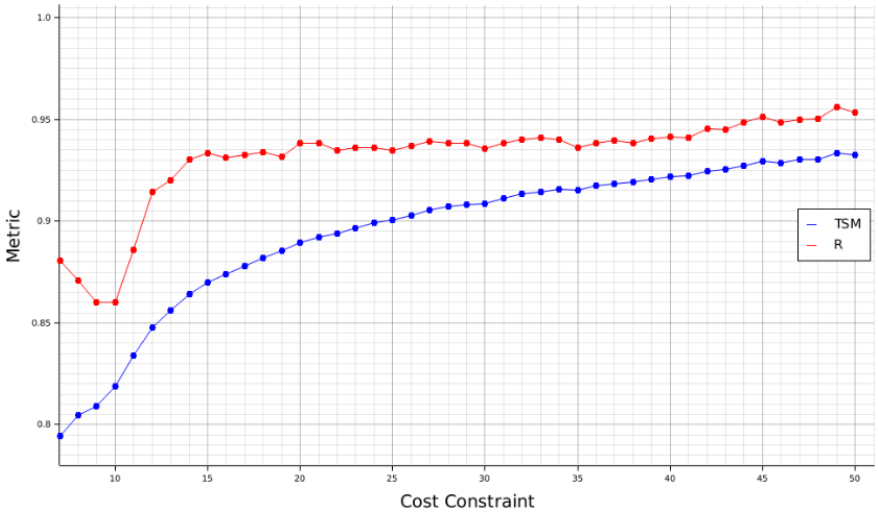

Figure 7. TSM and R under different total cost constraints.

3.7 Initial Credibility Score Value

Figure 8 and **Table 7** demonstrate TSM and R under varying initial values of S_i, where both metrics start at relatively high levels then steadily decrease as S_i increases from 3 to 60.

Table 7. TSM and R under different initial credibility score values.

S_i	TSM	R	S_i	TSM	R	S_i	TSM	R
3	0.8938	0.9420	23	0.8788	0.9237	43	0.8744	0.9166
4	0.8907	0.9381	24	0.8775	0.9221	44	0.8763	0.9208
5	0.8892	0.9380	25	0.8773	0.9202	45	0.8774	0.9224
6	0.8871	0.9342	26	0.8795	0.9261	46	0.8763	0.9198
7	0.8856	0.9324	27	0.8773	0.9207	47	0.8759	0.9186
8	0.8826	0.9263	28	0.8774	0.9229	48	0.8753	0.9168
9	0.8831	0.9289	29	0.8771	0.9211	49	0.8749	0.9171
10	0.8829	0.9291	30	0.8758	0.9174	50	0.8757	0.9185
11	0.8811	0.9261	31	0.8770	0.9185	51	0.8753	0.9181
12	0.8826	0.9282	32	0.8772	0.9222	52	0.8759	0.9172
13	0.8809	0.9257	33	0.8767	0.9187	53	0.8750	0.9172
14	0.8815	0.9281	34	0.8758	0.9208	54	0.8767	0.9232
15	0.8816	0.9287	35	0.8776	0.9209	55	0.8765	0.9182
16	0.8790	0.9218	36	0.8766	0.9201	56	0.8740	0.9147
17	0.8789	0.9239	37	0.8763	0.9177	57	0.8744	0.9173
18	0.8782	0.9219	38	0.8755	0.9202	58	0.8763	0.9218
19	0.8794	0.9246	39	0.8755	0.9169	59	0.8743	0.9156
20	0.8772	0.9180	40	0.8748	0.9169	60	0.8737	0.9133
21	0.8778	0.9214	41	0.8766	0.9212			
22	0.8793	0.9249	42	0.8757	0.9178			

When S_i is between 3 and 8, both metrics decrease rapidly. TSM moves from 0.8938 to 0.8826 and R moves from 0.9420 to 0.9263. From $S_i = 8$ to 30, the decline is more gradual and then plateaus at about 0.8750 for TSM and about 0.92 for R. From S_i = 30 to 60, the curves are stable with small oscillations that are linked to the master agent performing minimum allocations before the system is free from collusion. These patterns imply that a higher initial credibility score value makes it slightly more difficult to distinguish honest agents from collusive agents, resulting in a gradual decline in system performance.

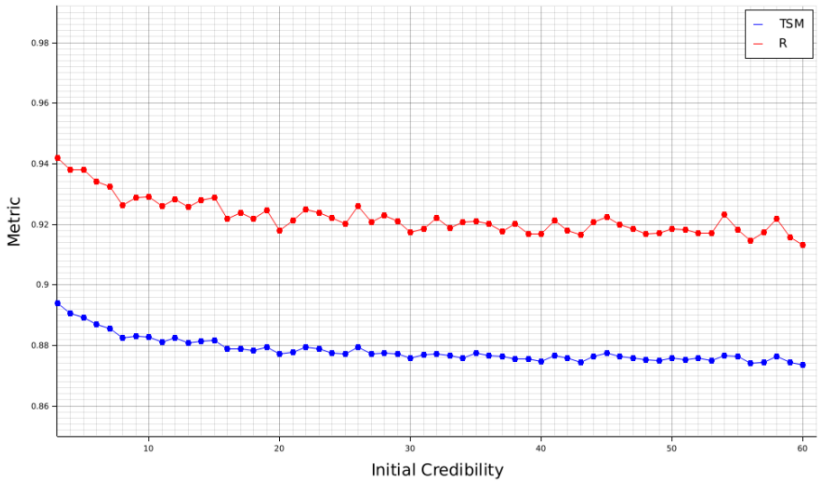

Figure 8. TSM and R under different initial credibility score values.

3.8 Cost Efficiency Bonus

The cost efficiency bonus (λ) gives the master agent a bonus for TSM for allocating less than the cost constraint (C^*), rewarding for achieving success with fewer agents. **Table 8** and **Figure 9** demonstrate the decrease of TSM and R as λ is varied from 0.05 to 0.9999 in increments of 0.05, showing a clear tradeoff between cost efficiency and performance. TSM drops more sharply while R declines more gradually and then plateaus at around 0.92. When λ is small, the cost efficiency has a little impact on TSM so the master agent chooses to allocate larger amounts of spotter and resource agents. Having more agents is better for finding collusive agents, leading to a higher chance of a successful task execution thus yielding higher TSM and R. As λ grows, the policy shifts to compact allocations to satisfy a higher cost efficiency, which reduces the number of agents allocated. This lowers the number of agents needed for a majority vote, leaving the system more open to collusion. Overall, lower λ raises both TSM and R by prioritizing correctness over efficiency, while higher λ sacrifices TSM and R for cost efficiency.

Table 8. TSM and R under cost efficiency bonus.

λ	TSM	R	λ	TSM	R
0.0500	0.9619	0.9693	0.5500	0.8466	0.9270
0.1000	0.9362	0.9483	0.6000	0.8398	0.9237
0.1500	0.9235	0.9463	0.6500	0.8324	0.9234
0.2000	0.9090	0.9368	0.7000	0.8256	0.9218
0.2500	0.8982	0.9346	0.7500	0.8212	0.9232
0.3000	0.8882	0.9355	0.8000	0.8159	0.9235
0.3500	0.8795	0.9320	0.8500	0.8100	0.9205
0.4000	0.8699	0.9276	0.9000	0.8044	0.9175
0.4500	0.8618	0.9289	0.9500	0.7983	0.9164
0.5000	0.8536	0.9274	0.9999	0.7926	0.9146

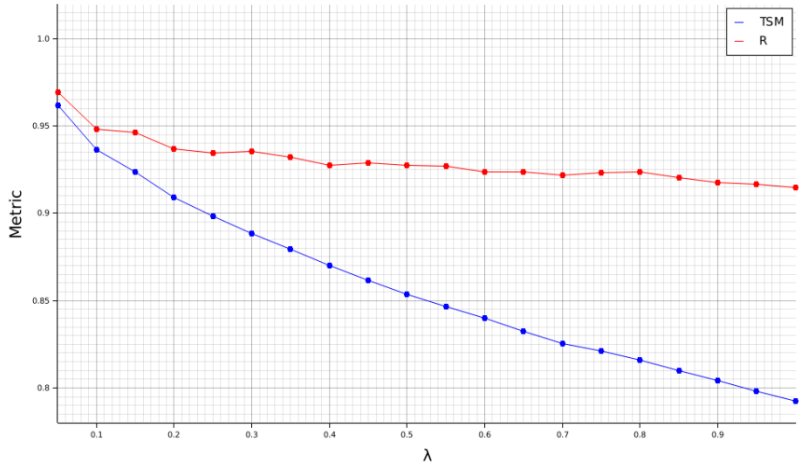

Figure 9. TSM and R under varying cost efficiency bonus.

4. Conclusion and Future Work

This work contributes by developing a multi-agent defense framework that systematically integrates a Q-learning based cost-aware agent allocation policy, a dual-stage spot-checking mechanism, and a strategic credibility adjustment policy. Multiple extensive simulations show the learning policy consistently achieves high task success probability under diverse conditions while staying cost effective. Compared with the recent work in Oikarinen and Xing (2025) using the rule driven systemic allocation, which exploits the full cost constraint and adapts slowly under heavy collusion, the proposed defense system is adaptive and implements cost efficient allocations. It allocates more spotter agents when there is heavy collusion, then lowers the spotter agent allocation as the number of collusive agents decreases. This yields a higher TSM and an acceptable R while avoiding unnecessary spending for agent allocations. Experimental results reveal the following insights:

1) Higher honest agent reliability improves both TSM and R, as it ensures that honest agents consistently generate correct outputs, thereby minimizing the impact of collusive agents.
2) As the total cost constraint increases, both TSM and R improve, but the improvements plateaus when an optimal allocation is reached for the total number of agents.
3) Lower efficiency bonus λ yields higher TSM and slightly higher R, as the reward places less weight on cost efficiency and more on correctness, guiding the master agent to use more of the cost constraint while allocating agents.
4) Higher collusion levels (i.e., more collusive agents in the agent pool) lower both TSM and R. The master agent counters this by allocating more agents in early episodes to remove more collusive agents faster.
5) A higher resource agent cost reduces both TSM and R because fewer work agents make it harder to achieve a majority vote. A lower spotter agent cost provides a small increase in TSM.
6) Harsher credibility penalties for incorrect outputs and higher rewards for correct output both improve TSM and R with larger penalties gaining more.
7) Higher initial credibility reduces both TSM and R because collusive agents take longer to be filtered out. Since all agents start with higher credibility, it becomes more difficult to distinguish collusive agents from honest agents, allowing collusive agents to have more influence on voting outcomes before their credibility is reduced.

This work assumes perfect spotter agents. This assumption will be relaxed by considering false negative and false positive errors in spot checking. Honest agent reliability is modeled as a fixed probability. Future research will relax this assumption by modeling reliability using a time-to-failure distribution such as the exponential or Weibull distribution. Another direction is to explore a more expressive reinforcement learning algorithm than Q-learning like proximal policy optimization or deep Q-learning to better handle larger state spaces and improve training stability. Additional future research includes testing alternative trust models (e.g. Beta reputation, Dirichlet reputation) and voting mechanisms (e.g. plurality voting, weighted voting), scaling experiments using larger agent pools, and implementing the defense system in a blockchain consensus scenario (Lei et al., 2025) or an agentic AI environment (Xing and Lin, 2025). These extensions can further improve both resilience and efficiency for voting-based systems under collusion attacks.

Conflicts of Interest
The authors confirm that there is no conflict of interest to declare for this publication.

Acknowledgments
This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors. The authors would like to thank the editor and anonymous reviewers for their comments that help improve the quality of this work.

AI Disclosure
The authors declare that no assistance is taken from generative AI to write this article.

References

Aguiar, N., Venkitasubramaniam, P., & Gupta, V. (2022). Data-driven contract design for multi-agent systems with collusion detection. *IEEE Signal Processing Letters*, *29*, 1002-1006. https://doi.org/10.1109/LSP.2022.3163691.

Avizienis, A., Gilley, G.C., Mathur, F.P., Rennels, D.A., Rohr, J.A., & Rubin, D.K. (1971). The STAR (self-testing and repairing) computer: an investigation of the theory and practice of fault-tolerant computer design. *IEEE Transactions on Computers*, *C-20*(11), 1312-1321. https://doi.org/10.1109/T-C.1971.223133.

Bennis, M., & Niyato, D. (2010). A Q-learning based approach to interference avoidance in self-organized femtocell networks. In *2010 IEEE Globecom Workshops* (pp. 706-710). IEEE. Miami, FL, USA. https://doi.org/10.1109/GLOCOMW.2010.5700414.

Bertrand, Q., Duque, J., Calvano, E., & Gidel, G. (2023). Q-learners can provably collude in the iterated prisoner's dilemma. *Computer Science and Game Theory.* arXiv preprint arXiv:2312.08484.

Bonjour, T., Aggarwal, V., & Bhargava, B. (2022). Information theoretic approach to detect collusion in multi-agent games. In *Proceedings of the 38th Conference on Uncertainty in Artificial Intelligence* (pp. 223-232). PMLR. Eindhoven, The Netherlands.

Chaisawat, S., & Vorakulpipat, C. (2020). Fault-tolerant architecture design for blockchain-based electronics voting system. In *2020 17th International Joint Conference on Computer Science and Software Engineering* (pp. 116-121). IEEE. Bangkok, Thailand. https://doi.org/10.1109/JCSSE49651.2020.9268264.

Chen, L., & Avizienis, A. (1978). N-version programming: a fault-tolerance approach to reliability of software operation. In *Proceedings of the 8th IEEE International Symposium on Fault-Tolerant Computing* (Vol. 1, pp. 3-9). IEEE. USA.

Chen, Y., Wang, Z., Min, Y., & Liu, Z. (2025). Decentralized-voting-based federated learning framework for lightweight node selection in edge collaborative IoT. *IEEE Internet of Things Journal*, *12*(12), 20272-20287.

Ding, D., Fan, X., Zhao, Y., Kang, K., Yin, Q., & Zeng, J. (2020). Q-learning based dynamic task scheduling for energy-efficient cloud computing. *Future Generation Computer Systems*, *108*, 361-371. https://doi.org/10.1016/j.future.2020.02.018.

Gaur, D., Bali, S., Gunasekaran, A., Joshi, N., Chaudhary, K., & Bali, V. (2025). A multi-criteria decision-making framework for blockchain technology adoption in smart healthcare. *International Journal of Mathematical, Engineering and Management Sciences, 10*(5), 1476-1496. https://doi.org/10.33889/IJMEMS.2025.10.5.070.

Iqbal, S., Qureshi, A.N., Alhussein, M., Aurangzeb, K., Mahmood, A., & Azzuhri, S.R.B. (2025). Dynamic selectout and voting-based federated learning for enhanced medical image analysis. *Machine Learning: Science and Technology*, *6*(1), 015010. https://doi.org/10.1088/2632-2153/ada0a6.

Ivanov, R., Pajic, M., & Lee, I. (2016). Attack-resilient sensor fusion for safety-critical cyber-physical systems. *ACM Transactions on Embedded Computing Systems*, *15*(1), 1-24. https://doi.org/10.1145/2847418.

Jumaah, M.A., Ali, Y.H., & Rashid, T.A. (2025). Efficient Q-learning hyperparameter tuning using FOX optimization algorithm. *Results in Engineering*, *25*, 104341. https://doi.org/10.1016/j.rineng.2025.104341.

Lei, T., Zhang, Q., Qiu, W., Zheng, H., Miao, S., Jie, W., Zhu, J., Dong, J., & Zheng, Z. (2025). An enhanced DPoS consensus mechanism using quadratic voting in Web 3.0 ecosystem. *Blockchain*, *3*(1), 1-2. https://doi.org/10.55092/blockchain20250001.

Levitin, G., Xing, L., & Dai, Y. (2019). Optimal spot-checking for collusion tolerance in computer grids. *IEEE Transactions on Dependable and Secure Computing*, *16*(2), 301-312. https://doi.org/10.1109/TDSC.2017.2690293.

Levitin, G., Xing, L., Johnson, B.W., & Dai, Y. (2018). Optimization of dynamic spot-checking for collusion tolerance in grid computing. *Future Generation Computer Systems*, *86*, 30-38. https://doi.org/10.1016/j.future.2018.01.049.

Li, H., Hui, H., & Zhang, H. (2023). Decentralized energy management of microgrid based on blockchain-empowered consensus algorithm with collusion prevention. *IEEE Transactions on Sustainable Energy*, *14*(4), 2260-2273. https://doi.org/10.1109/TSTE.2023.3258452.

Li, X., Wang, S., Zeng, S., Wu, Y., & Yang, Y. (2024). A survey on LLM-based multi-agent systems: workflow, infrastructure, and challenges. *Vicinagearth*, *1*(1), 9. https://doi.org/10.1007/s44336-024-00009-2.

Miao, B., Xu, Y., Wang, J., & Zhang, Y. (2024). Dc-bvm: dual-channel information fusion network based on voting mechanism. *Biomedical Signal Processing and Control*, *94*, 106248. https://doi.org/10.1016/j.bspc.2024.106248.

Mohammed, H., & Rashid, T. (2023). FOX: a FOX-inspired optimization algorithm. *Applied Intelligence*, *53*(1), 1030-1050. https://doi.org/10.1007/s10489-022-03533-0.

Motwani, S., Baranchuk, M., Strohmeier, M., Bolina, V., Torr, P., Hammond, L., & de Witt, C.S. (2024). Secret collusion among AI agents: multi-agent deception via steganography. *Advances in Neural Information Processing Systems*, *37*, 73439-73486. https://doi.org/10.52202/079017-2336.

Nie, J., & Haykin, S. (1999). A Q-learning-based dynamic channel assignment technique for mobile communication systems. *IEEE Transactions on Vehicular Technology*, *48*(5), 1676-1687. https://doi.org/10.1109/25.790549.

Nordmann, L., & Pham, H. (1999). Weighted voting systems. *IEEE Transactions on Reliability*, *48*(1), 42-49. https://doi.org/10.1109/24.765926.

Oikarinen, N., & Xing, L. (2025). A heterogeneous multi-agent colluding attack defense system. *International Journal of Mathematical, Engineering and Management Sciences*, *10*(6), 1640-1657. https://doi.org/10.33889/IJMEMS.2025.10.6.078.

Owoputi, R., & Ray, S. (2022). Security of multi-agent cyber-physical systems: a survey. *IEEE Access*, *10*, 121465-121479. https://doi.org/10.1109/ACCESS.2022.3223362.

Pang, X., Fang, X., Yu, Y., Zheng, Z., & Li, H. (2025). Optimal scheduling method for electric vehicle charging and discharging via Q-learning-based particle swarm optimization. *Energy*, *316*, 134611.

Parhami, B. (1994). Voting algorithms. *IEEE Transactions on Reliability*, *43*(4), 617-629.

Pilar, B. (2025). InceptionV3-driven multiclassifier voting system for watermark classification. In *2025 International Conference on Computational, Communication and Information Technology* (pp. 471-476). IEEE. Indore, India. https://doi.org/10.1109/ICCCIT62592.2025.10928146.

Singh, D., & Singh, B. (2022). Feature wise normalization: an effective way of normalizing data. *Pattern Recognition*, *122*, 108307. https://doi.org/10.1016/j.patcog.2021.108307.

Staab, E., & Engel, T. (2009). Collusion detection for grid computing. In *2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid* (pp. 412-419). IEEE. Shanghai, China. https://doi.org/10.1109/CCGRID.2009.12.

Sutton, R.S., & Barto, A.G. (2018). *Reinforcement learning: an introduction*. Cambridge: MIT press. London, England.

Tasci, E., Uluturk, C., & Ugur, A. (2021). A voting-based ensemble deep learning method focusing on image augmentation and preprocessing variations for tuberculosis detection. *Neural Computing and Applications*, *33*(22), 15541-15555. https://doi.org/10.1007/s00521-021-06177-2.

Tassa, T., Grinshpoun, T., & Yanai, A. (2019). A privacy preserving collusion secure DCOP algorithm. *Cryptography and Security*. arXiv preprint arXiv:1905.09013.

Verma, S., Chandra, G., & Yadav, D. (2025). LVCA: an efficient voting-based consensus algorithm in private Blockchain for enhancing data security. *Peer-to-Peer Networking and Applications*, *18*(2), 88.

Wang, Q., Li, W., & Mohajer, A. (2024). Load-aware continuous-time optimization for multi-agent systems: toward dynamic resource allocation and real-time adaptability. *Computer Networks*, *250*, 110526.

Wang, S., Qu, X., Hu, Q., Wang, X., & Cheng, X. (2023). An uncertainty-and collusion-proof voting consensus mechanism in blockchain. *IEEE/ACM Transactions on Networking*, *31*(5), 2376-2388.

Wang, W., An, B., & Jiang, Y. (2018). Optimal spot-checking for improving evaluation accuracy of peer grading systems. *Proceedings of the AAAI Conference on Artificial Intelligence*, *32*(1), 833-840.

Wang, W., An, B., & Jiang, Y. (2020). Optimal spot-checking for improving the evaluation quality of crowdsourcing: application to peer grading systems. *IEEE Transactions on Computational Social Systems*, *7*(4), 940-955.

Watkins, C.J.C.H., & Dayan, P. (1992). Q-learning. *Machine Learning*, *8*(3), 279-292.

Xing, L., & Lin, J. (2025). Looking forward: challenges and opportunities in agentic AI reliability. In *Generative and Agentic AI Reliability: Architectures, Challenges, and Trust for Autonomous Systems*. Springer Nature, preprint. https://doi.org/10.48550/arXiv.2511.11921.

Yang, Z., Hu, X., Li, Y., Liang, M., Wang, K., Wang, L., Tang, H., Guo, S. (2025). A Q-learning-based improved multi-objective genetic algorithm for solving distributed heterogeneous assembly flexible job shop scheduling problems with transfers. *Journal of Manufacturing Systems*, *79*, 398-418.

Zhang, H., & Zhou, Y. (2023). A neural network-based weighted voting algorithm for multi-target classification in WSN. *Sensors*, *24*(1), 123. https://doi.org/10.3390/s24010123.

Zhao, N., Shi, M., Zhao, X., Zong, G., & Zhang, H. (2024). Distributed adaptive sampled-data security tracking control for uncertain heterogeneous multi-agents systems under DoS attacks. *IEEE Transactions on Green Communications and Networking*, *8*(4), 1385-1397. https://doi.org/10.1109/TGCN.2024.3381346.