Trust-based Consensus in Multi-Agent Reinforcement Learning Systems

Ho Long Fung[1], **Victor-Alexandru Darvariu**[1], **Stephen Hailes**[1], **Mirco Musolesi**[1,2]
{ho.fung.20, v.darvariu, s.hailes, m.musolesi}@ucl.ac.uk
[1]University College London [2]University of Bologna

Abstract

An often neglected issue in multi-agent reinforcement learning (MARL) is the potential presence of unreliable agents in the environment whose deviations from expected behavior can prevent a system from accomplishing its intended tasks. In particular, consensus is a fundamental underpinning problem of cooperative distributed multi-agent systems. Consensus requires different agents, situated in a decentralized communication network, to reach an agreement out of a set of initial proposals that they put forward. Learning-based agents should adopt a protocol that allows them to reach consensus despite having one or more unreliable agents in the system. This paper investigates the problem of unreliable agents in MARL, considering consensus as a case study. Echoing established results in the distributed systems literature, our experiments show that even a moderate fraction of such agents can greatly impact the ability of reaching consensus in a networked environment. We propose Reinforcement Learning-based Trusted Consensus (RLTC), a decentralized trust mechanism, in which agents can independently decide which neighbors to communicate with. We empirically demonstrate that our trust mechanism is able to handle unreliable agents effectively, as evidenced by higher consensus success rates.

1 Introduction

A *cooperative multi-agent system* (MAS) is a system composed of multiple autonomous entities, known as *agents*, which collaborate within a shared environment to solve tasks in order to improve their joint welfare (Dafoe et al., 2020). A largely neglected yet fundamental issue in the multi-agent reinforcement learning (MARL) literature is the potential presence of *unreliable* agents within the environment. Indeed, dealing with unreliability is of essential importance for building real world MARL systems. An agent might be unreliable due to node and/or transmission failure, or simply because it exchanges incorrect information. Identifying the exact cause of unreliability is difficult and often impossible a priori.

A general class of cooperative tasks that is susceptible to unreliable agents is *consensus*. Consensus is indeed an essential element of the design of any distributed system, including multi-agent systems. It is one of the classic problems in multi-agent and distributed systems, which has fascinated generations of scientists from the foundations of the field (Lamport et al., 1982; Lamport, 1998) due to its theoretical elegance and vast applicability to a series of practical problems (Ongaro & Ousterhout, 2014; Olfati-Saber et al., 2007; Barrat et al., 2008). Consensus deals with reaching an agreement among agents that put forward different proposals. A *consensus protocol* can be defined as a procedure that agents follow in order to reach agreement successfully. A protocol can be hardcoded, where each agent executes an algorithm written by an expert (e.g., in the distributed systems literature (Cachin et al., 2011)). Otherwise, the protocol is *emergent* if it arises from agents learning how to achieve consensus through repeated interactions. Consensus problems are often studied in decentralized settings, where agents can only make local observations of the environment, act independently of each other and interact solely by communicating over a network (Lamport et al., 1982;

Pease et al., 1980; Cachin et al., 2011; Coulouris et al., 2012). These limitations make the problem of reaching consensus nontrivial. A reason is the lack of a central coordinator that can aggregate and distribute values to and from all agents in the system. Another reason is the existence of failed or unreliable agents in the network that deviate from their expected behavior. Decentralization renders failures hard to detect since this cannot be performed based on direct observation.

In real-world scenarios, the occurrence of unexpected failures is often an integral property of the environment (Schroeder & Gibson, 2010; Lianza & Snook, 2020). Suppose that a MAS adopts a consensus protocol that assumes agents are always reliable. In cases where agents fail unexpectedly, this may lead to errors in message transmission or performing local computations. Since the protocol cannot accommodate for these failures, agents will assume that the rest of the system is still reliable and execute the protocol as usual, using incomplete or incorrect information, which may severely impact the system's performance. Therefore, to reach consensus reliably, agents must adopt a protocol that can continue operating at an acceptable level despite one or more failures in the system. One way to deal with unreliable agents is through the introduction of *trust mechanisms*, which have been widely studied in the distributed systems and multi-agent systems literature (e.g., (Abdul-Rahman & Hailes, 1998; Ramchurn et al., 2004)). Agents can try to quantify the trustworthiness of other agents based on past interactions and data about them.

This paper investigates the problem of unreliable agents in MARL, considering the problem of distributed consensus as a case study. We start by assuming a network of reliable agents with a basic communication model, then introduce unreliable agents and observation noise and study their impact on achieving consensus. In line with classic results in distributed systems (e.g., Lamport et al. (1982); Pease et al. (1980)), we find that unreliable agents greatly impact the ability of decentralized learning-based setups to reach consensus. To mitigate this issue, we introduce the notion of trust by equipping RL agents with a learnable trust mechanism, which we refer to as *Reinforcement Learning-based Trusted Consensus (RLTC)*. Experiments show that RLTC increases the consensus success rate of the system when compared to a setting without the trust mechanism (i.e., in which agents assume all other agents are reliable), generalizing to various types of failure models. We also show that RLTC is able to scale as the number of agents increases, demonstrating practical applicability of the proposed mechanism to real-world scenarios.

2 Related Work

Emergent communication protocols in MARL. In the last decade, there has been a significant amount of work on deep multi-agent reinforcement learning for cooperative multi-agent tasks, such as traffic junctions (Sukhbaatar et al., 2016), multi-robot warehouses (Christianos et al., 2020), cooperative navigation and predator-prey games (Lowe et al., 2017). Differentiable communication mechanisms have been introduced to allow deep agents to choose what and to whom to communicate (Foerster et al., 2016; Sukhbaatar et al., 2016; Das et al., 2019; Rangwala & Williams, 2020; Zhang et al., 2020b). Progress is also being made in the design of *emergent communication protocols*, where agents learn to associate communication symbols with perceptual input and actions by solving cooperative downstream tasks (Mordatch & Abbeel, 2017; Lazaridou et al., 2018; Bouchacourt & Baroni, 2018; Graesser et al., 2019). However, this body of work often assumes that all agents are fully functional throughout the lifetime of the system. In our work, we instead investigate if RL agents can learn to adapt to the presence of unreliable agents.

Adversarial attacks in (MA)RL. Several works have studied the design of RL policies that are robust to adversarial attacks in a single-agent setting. Pinto et al. (2017) focused on continuous control environments in which an adversary is allowed to apply disturbances to the agent's action, demonstrating that more robust policies can be obtained even in the absence of the adversary at test time. Zhang et al. (2020a) considered a threat model in which the observations received by the agent are adversarially perturbed. In the MARL literature, Blumenkamp & Prorok (2021) showed that, in environments that are not fully cooperative, self-interested agents are able to learn to construct messages that disrupt the cooperative agents. Xue et al. (2022) proposed to train a model that recovers the true message from the malicious message, which relies on the assumption that the message is perturbed (rather than replaced altogether). Sun et al. (2023) considered a multi-agent setting and proposed a technique for dealing with adversarial agents that relies on aggregating the received messages (by majority vote for the discrete action case and the median for each coordinate in the continuous case). This technique requires the assumption of a non-adversarial training phase for learning the message aggregation policy. In contrast, RLTC is based on an explicit and interpretable trust mechanism that improves performance over standard MARL even when unreliable agents are present at training time.

Consensus in complex networks. Interactive models have been used to study the collective behavior of agents in networked environments, such as social networks and particle systems in general. Examples include the Voter model (Barrat et al., 2008) and majority rule (Krapivsky & Redner, 2003), where neighboring agents can communicate and update their local state using a random mechanism. In our work, we use a discrete-time variant of the Voter model, where each agent can choose who to communicate with by sampling from a subset of trusted neighbors, which the agent can adjust over time.

Averaging consensus algorithms. There are other formulations of consensus applied to areas such as social influence networks (DeGroot, 1974), sensor networks (Olfati-Saber & Shamma, 2005) and vehicle formations (Fax & Murray, 2004). A relevant example is the class of *averaging algorithms*, where each node in a network updates its local scalar value by computing an average over its neighbors, potentially using different and/or randomized weights (Bullo, 2022). Properties of these algorithms, such as asymptotic convergence, have been studied with matrix theory and algebraic graph theory (Ren et al., 2005; Olfati-Saber et al., 2007). In contrast, our work uses RL, where each agent decides which neighbors to trust and aggregate values from, resulting in adaptive rather than pre-determined behavior.

Trust in distributed systems. Trust is important for supporting cooperation and coordination in multi-agent systems (Dafoe et al., 2020). Various formulations of trust have been studied in the distributed and multi-agent systems literature. Ways to quantify trust include discrete trust levels and scores, which are usually computed based on past interactions and data about other agents (Abdul-Rahman & Hailes, 1998; Sabater & Sierra, 2005). These can help an agent to decide if it is beneficial to cooperate with another agent. Models based on reputation and recommendations for gathering, aggregating and promoting ratings that approximate the trustworthiness of an agent have also been developed (Abdul-Rahman & Hailes, 1998; Castelfranchi & Falcone, 1998; Ramchurn et al., 2004; Sabater & Sierra, 2005; Cohen et al., 2019). Our work is related to the area of decentralized trust, where agents learn who to trust by interacting with others, rather than relying on a central authority (Abdul-Rahman & Hailes, 1998).

3 Problem Description

This section provides a formal definition of a consensus problem, including modeling assumptions such as the means of inter-agent communication and trust mechanisms.

Consensus definition. Consensus is reached when agents agree upon a common value out of a set of initial proposals. Denote the set of agents in the system as $\mathcal{N} = \{1, 2, \ldots, N\}$. They must agree upon a global value from a known set of candidates $\mathcal{D}$, which we assume is binary: $\mathcal{D} = \{0, 1\}$. We also define a predetermined ground-truth value as 1, while the incorrect value is 0. Initially, each agent $i \in \mathcal{N}$ proposes a value $v_i \in \mathcal{D}$ with independent probability p of being equal to 1 and 0 with probability $1 - p$, the latter of which we will refer to as the *noise parameter*. Then, agents can share information over a communication network and update their local values appropriately. Eventually, consensus is reached if all agents agree upon the true value, otherwise known as *global consensus*. However, due to the decentralized nature of our setup, where agents can only interact with their immediate neighbors, it is difficult for agents to determine if all other agents agreed on 1. Thus, we define an agent-specific success criterion, known as *local consensus*, which is when the agent and its neighbors in the network agree on 1.

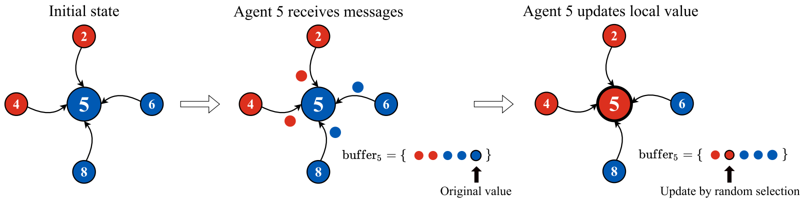

Initial state
2
4
5
6
8
Agent 5 receives messages
2
4
5
6
8
buffer$_5$ = { ● ● ● ● ● }
Original value
Agent 5 updates local value
2
4
5
6
8
buffer$_5$ = { ● ● ● ● ● }
Update by random selection

Figure 2: Communication mechanism from the perspective of agent 5. Agent 5 receives the values from its neighbors, then updates its local value by randomly selecting from the set of received values and its own. During each timestep, this is performed simultaneously by all agents.

Communication model. Agents are interconnected by a communication network. It is defined by an undirected, connected graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ with nodes $\mathcal{N}$, which correspond to agents, and bi-directional communication links $\mathcal{E}$ between nodes. For simplicity, we assume the communication network is a 2D square lattice. See Figure 1 for an illustration with $N = 9$ agents. Agents interact by performing consecutive message-passing rounds in lock-step. A round starts with each agent broadcasting its current value v_i to all of its neighbors $\mathrm{ne}(i)$. Then, upon receiving all incoming values into a message buffer denoted buffer_i, each agent updates its current value by randomly selecting a value in $\{v_i\} \cup \mathrm{buffer}_i$. See Figure 2 as an example of communication between an agent and its neighbors. This communication mechanism described is similar to opinion formation models which are used to simulate interactions in social networks (Barrat et al., 2008). They assume a population of agents that initially have contradictory opinions and can update their own opinions over time by interacting pairwise with other agents in the network. This is also similar to randomized averaging algorithms, where nodes can choose which values to aggregate by following a stochastic model (Bullo, 2022).

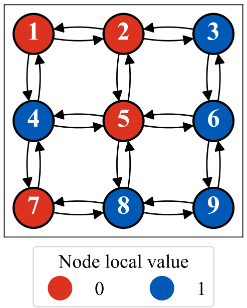


Figure 1: Communication network example with 9 agents, where nodes represent agents, arrows indicate communication links and colors are mapped to the agents' local values.

Failure models. In addition, we specify the *failure models* that describe how agents can deviate from expected behavior. Here, we assume that a faulty or unreliable agent may send the incorrect value 0 to its neighbors. This is shown in Figure 3. If an unreliable agent does this all the time, we refer to this as following a *Fixed* failure model. Another possibility is that the agent sends 0 or 1 uniformly at random, which we refer to as the *Random* failure model. We will investigate the extent of which our trust-based mechanism that we introduce next can generalize to both types of unreliability. In addition, Figure 6a illustrates an example network with a single unreliable Fixed agent in which, despite the simple failure model, consensus is not achieved due to the propagation of misinformation by the other agents. Note that agents cannot determine the reliability of other agents a priori due to the decentralized property of the system, hence learning is necessary.

4 Trust Mechanisms for Consensus

A potential solution to our consensus problem is through the notion of trust, as introduced in Section 2. To this end, we propose a simple decentralized trust mechanism that can be learned through MARL. In this section, we present its basic operation, then Section 5 will provide the reader with a more formal description of the proposed solution.

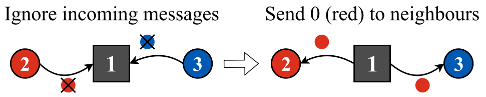


Figure 3: Example of an unreliable agent 1 (square) that always ignores messages from neighbors 2 and 3 and sends 0.

Each agent i maintains a binary score for each neighbor j, representing whether i trusts j. Denote $\mathrm{trusts}_i(j)$ as the trust score of agent i towards j and $\mathrm{trusts}_i(\cdot)$ as the array containing all trust scores

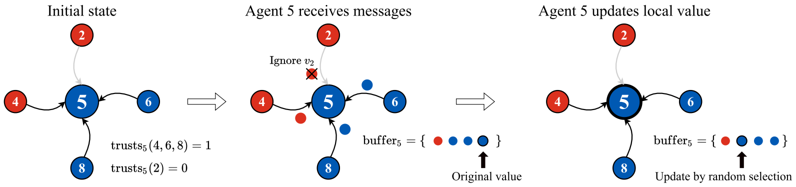

Initial state
Agent 5 receives messages
Agent 5 updates local value
2
4
5
6
8
Ignore v_2
$\text{trusts}_5(4,6,8) = 1$
$\text{trusts}_5(2) = 0$
$\text{buffer}_5 = \{$ ● ● ● ● $\}$
Original value
Update by random selection

Figure 4: Communication between agent 5 and its neighbors, but agent 5 is equipped with a trust mechanism. Agent 5 trusts neighbors 4, 6 and 8 but not agent 2. It updates its local value by randomly sampling from itself and its *trusted* neighbors only.

Algorithm 1: RLTC Consensus Protocol Episode Execution

```
Initialize each $v_i$ randomly to 1 with probability $p$, 0 otherwise.
Initialize all $\text{trusts}_i(j) = 1$.
for timestep $t = 1, 2, \dots, T$ do
    for $i \in \mathcal{N}$ do
        // Receive messages
        $\text{buffer}_i \leftarrow \{v_j \mid j \in \text{ne}(i) \wedge \text{trusts}_i(j) = 1\}$
    for $i \in \mathcal{N}$ do
        // Update local values
        $v_i \leftarrow \text{random}(\{v_i\} \cup \text{buffer}_i)$
    for $i \in \mathcal{N}$ do
        $j \leftarrow \pi^i(s_t^i)$;
        if $j \neq \emptyset$ then
            // Update trust score
            $\text{trusts}_i(j) \leftarrow \neg\text{trusts}_i(j)$
        // If training, do $Q$-learning update for agent $i$ (Eq. 1)
```

of that agent. If an agent i distrusts a neighbor j, or $\text{trusts}_i(j) = 0$, then it can choose to ignore all incoming messages from j.

In effect, the array $\text{trusts}_i(\cdot)$ defines the subset of neighbors from which agent i is allowed to sample values from during each timestep. Figure 4 demonstrates this from the perspective of one agent. Figure 5 illustrates an example of trust in a grid. For example, both agents 4 and 5 distrust each other ($\text{trusts}_4(5) = 0$ and $\text{trusts}_5(4) = 0$). As a result, both agents ignore each other's incoming messages, as indicated by the grayed out arrows from node 5 to 4 and vice versa. Agents 2 and 3 mutually trust each other, as shown by black arrows in both directions. Agent 1 distrusts Agent 4 (so Agent 1 ignores 4's messages, indicated by the grayed out *incoming* arrow into Agent 1 from Agent 4), but Agent 4 trusts Agent 1. This is an example of unreciprocated trust. Through continual interactions, each agent can update its trust scores in order to minimize the effects of unreliable agents and improve the chances of successful consensus.

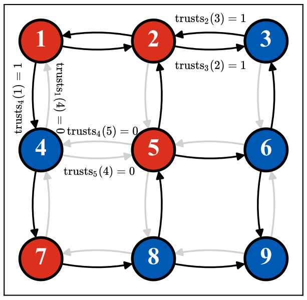


Figure 5: Communication grid example with 9 agents, in which not all agents trust each other.

5 MARL Model

Given the trust mechanism discussed in Section 4, we will describe how the agents can learn to update their trust scores. This is achieved by formulating the problem as a multi-agent decentralized Markov Decision Process, then learning the trust mechanism using Q-learning (Watkins & Dayan, 1992; Tan, 1993). The multi-agent MDP that we will use is characterized by a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R)$, containing the state space, action space, transition probability and reward function respectively. Each of these elements is described below. In addition, we assume a finite horizon of length T. Note

Name	Equation	Description
Success rate	$\frac{1}{N}\sum_{i=1}^{N}\mathbb{1}[v_i = v_{\text{true}}]$	Fraction of reliable agents with the correct value.
Average trust rate	$\frac{1}{N}\sum_{i\in\mathcal{N}}\frac{1}{\lvert\text{ne}(i)\rvert}\sum_{j\in\text{ne}(i)}\text{trusts}_i(j)$	Average fraction of neighbors, reliable or otherwise, that each reliable agent trusts.
Mutual trust rate	$\frac{1}{\lvert\mathcal{E}\rvert}\sum_{i,j\in\mathcal{E}}\mathbb{1}[\text{trusts}_i(j)\wedge\text{trusts}_j(i)]$	Fraction of mutually-trusting reliable agents. 0 if no reliable agents are adjacent in the network.
Average trust accuracy	$\frac{1}{N}\sum_{i\in\mathcal{N}}\frac{1}{\lvert\text{ne}(i)\rvert}\sum_{j\in\text{ne}(i)}\mathbb{1}[\text{trusts}_i(j)=\mathbb{1}[j\in\mathcal{N}]]$	Fraction of correct trust scores assigned to neighbors on average, i.e., the agent assigns trust score 1 for reliable neighbors and 0 for unreliable ones.

Table 1: Performance metrics used in the evaluation.

that each agent i in the MDP corresponds to a *reliable* node in the network only. Unreliable nodes follow a pre-defined behavior as described in Section 3. They do not have a trust mechanism and are not trained, thus are excluded from the MDP. Figure 6b illustrates an example of episodic execution with 9 agents.

States. $\mathcal{S} = \prod_{i\in\mathcal{N}}\mathcal{S}^i$ is the global state space, which is composed of the local state spaces $\mathcal{S}^i$ per agent i. An agent's local state s^i corresponds to its trust array $\text{trusts}_i(\cdot)$ as described in Section 4. In other words, s^i maps neighbors to trust scores: $s^i : \text{ne}(i) \mapsto \{0, 1\}$. For example, Agent 1 in Figure 5 has $\text{trusts}_1(2) = 1$ and $\text{trusts}_1(4) = 0$, so $s^1 = \{(2, 1), (4, 0)\}$. Agent 5's local state is $s^5 = \{(2, 0), (4, 0), (6, 0), (8, 1)\}$. At the start of each episode, all entries of all trust arrays are initialized to 1 (e.g., $s^5 = \{(2, 1), (4, 1), (6, 1), (8, 1)\}$ at $t = 0$). Note that local states only consider trust scores and are independent of the agent's current local value (0 or 1).

Actions. $\mathcal{A} = \prod_{i\in\mathcal{N}}\mathcal{A}^i$ is the joint action space composed of the individual local action space $\mathcal{A}^i$ of each agent i. A local action a^i corresponds to agent i toggling its trust score of a specific neighbor or not toggling any score at all. So, $\mathcal{A}^i = \{\emptyset\} \cup \text{ne}(i)$, where $\emptyset$ is a no-op action. Concretely, if $a^i = j$ then $\text{trusts}_i(j) \leftarrow \neg\text{trusts}_i(j)$. For example, in Figure 6b between $t = 0$ and $t = 1$, agent 1 toggles its trust score of neighbor 2 from $\text{trusts}_1(2) = 1$ to 0 by selecting action $a^1 = 2$. Agent 4 did not modify its trust scores, so $a^4 = \emptyset$. Due to the decentralized nature of our setup, agents select actions a^i independently, following their local policy $\pi^i(s^i)$ and independently of other agents.

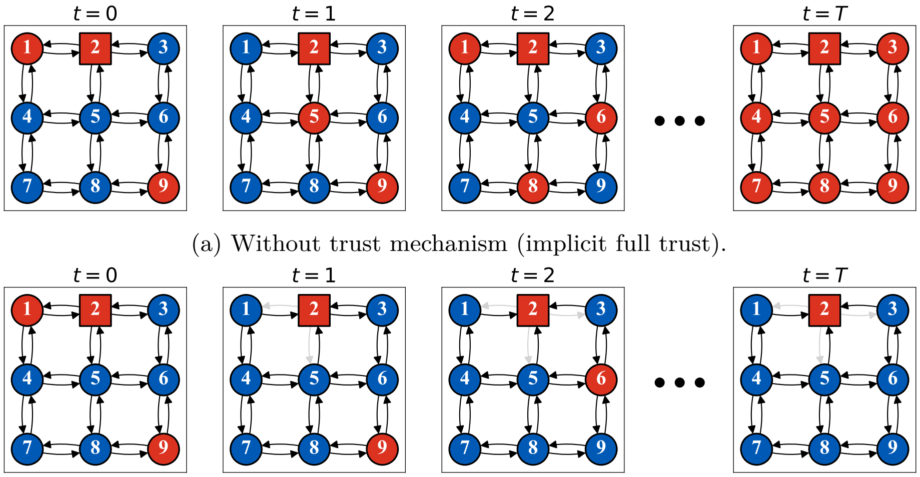
Transitions. P is the transition function: $P(s'|s, \{a^i\}_{i\in\mathcal{N}})$ is the probability of moving from global state $s \in \mathcal{S}$ to state $s' \in \mathcal{S}$ when each agent i executes some action a^i. The global state is the union of the trust score arrays of all agents in the system after they individually perform local actions (toggling or doing nothing). The state transitions are deterministic, only depending on the agents' actions. See Figure 6b for examples of consecutive global state transitions, but ignore the agents' local values (red or blue) because the MDP does not consider this information.

Reward and consensus optimization objective. The goal of the trust mechanism is to ensure that consensus can still be attained despite disruptions caused by unreliable nodes in the network. Each reliable agent must update its trust scores so that it reaches agreement with its neighbors correctly. To achieve this, it must discover a policy π^i that allows it to perform appropriate updates to its trust array. Thus, we define the reward function R^i in terms of the local success criteria for consensus described in Section 3, which is appropriate for our decentralized setup where agents can only interact directly with their neighbors. Each agent i is assigned reward $R^i = +1$ if it correctly agrees with all neighbors: $\forall j \in \{i\}\cup\text{ne}(i) :\ v_j = 1$. If this is not satisfied, then $R^i = -1$. The global objective is to find an optimal joint policy $\pi = \langle\pi^1, \ldots, \pi^N\rangle$ that maximizes the joint action-value function $Q^\pi(s, a) = \mathbb{E}\left[\sum_{k=t+1}^{T}\gamma^{k-t-1}\sum_{i\in\mathcal{N}} R_k^i | s_t = s, a_t = a\right]$, where s is the joint state, a is the joint action, T is the horizon and $\gamma \in [0, 1)$ is the discount factor.

***Q*-learning update.** Due to the decentralized setup, we will optimize the above objective using independent Q-learning (Tan, 1993), where each agent learns according to its own actions and local state. The update rule for agent $i \in \{1, \dots N\}$ is as follows:

$$Q^i(s_t^i, a_t^i) \leftarrow Q^i(s_t^i, a_t^i) + \alpha \left(\left[R_t^i + \gamma \max_{a \in \mathcal{A}^i} Q^i(s_{t+1}^i, a) \right] - Q^i(s_t^i, a_t^i) \right) \tag{1}$$

where Q^i is the Q-function of agent i, α is the learning rate and γ is the discount factor. The update uses states and actions local to agent i, i.e., s^i and a^i. Agents use the ϵ-greedy policy for training and the greedy policy for evaluation.



(a) Without trust mechanism (implicit full trust).

(b) With trust mechanism (agents 1, 5 and 3 distrust agent 2).

Figure 6: An example scenario with 9 agents with one unreliable agent labeled 2. Timesteps 0, 1, 2 and $T = 30$ of an episode are shown. Figure 6a does not have the trust mechanism and results in failure, because the incorrect value 0 (red) has spread to all agents. Figure 6b has the trust update mechanism. Reliable agents 1 and 5 toggle their trust scores of agent 2 to 0 at $t = 1$, then agent 3 does the same at $t = 2$. Here, consensus is successful, because agent 2 is effectively isolated from the rest of the network, and all reliable agents agree on 1 (blue).

Summary of consensus protocol execution. The full execution of our consensus protocol is detailed in Algorithm 1. To aid understanding, an example is illustrated in Figure 6b. At the beginning, the communication graph contains both reliable and unreliable agents. In the example, only agent 2 is unreliable. Reliable agents are trained, and unreliable agents have fixed behavior as described in Section 3. Then, the consensus protocol consists of both the trust mechanism (set up as an MDP above) and communication (sending and updating local values, which are not part of the MDP). The protocol executes over a finite episode with length T. At the start of each episode, all trust scores are initialized to 1, as indicated by all black arrows in Figure 6b. Each reliable agent's local value is randomly set to 0 with probability $1-p$, 1 otherwise, while the unreliable agents always start with 0. In the example, reliable agents 1 and 9 start with 0 while the others have 1, while unreliable agent 2 has 0. Then, each timestep consists of three phases: agents receive the incoming values of trusted neighbors, update their local values, and finally update their trust scores (decided by the agents' local action policies). The episode terminates after the final timestep T is reached.

6 Experiments

Our goal is to determine whether the proposed decentralized trust mechanism can increase the chances of successful consensus despite the presence of unreliable agents, for either the *Fixed* or

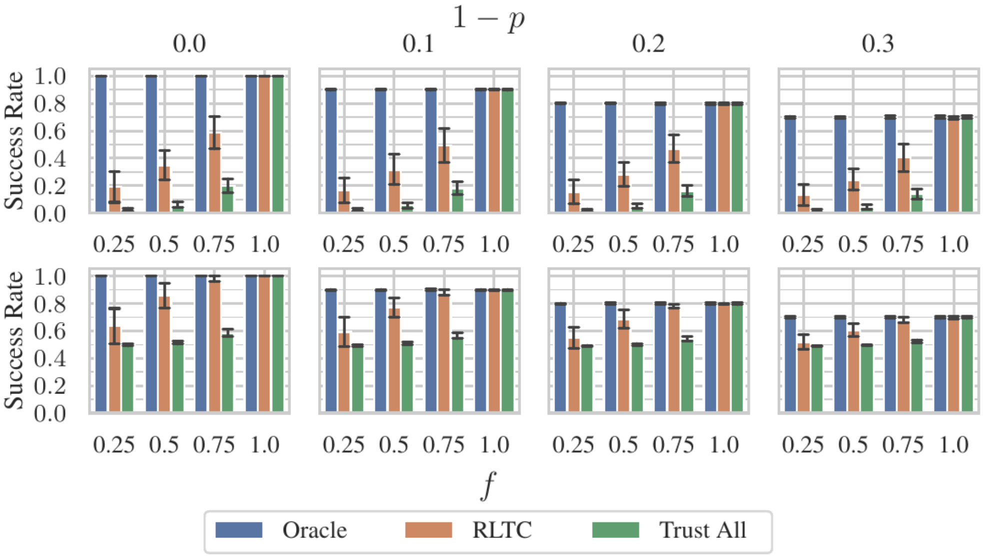


Figure 7: Success rates for different fractions of reliable agents f and noise values $1 - p$ (16 agents). Each bar height and error bar represent the mean and 1 standard deviation respectively wrt. 30 runs. Top and bottom plots are for the *Fixed* and *Random* failure models respectively.

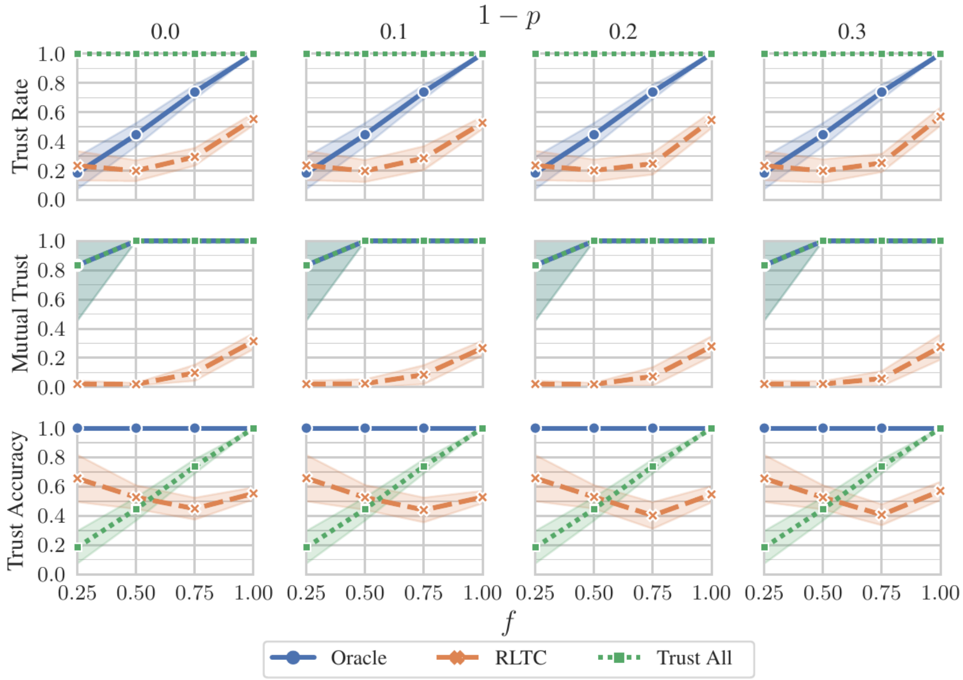


Figure 8: f against trust metrics for fixed $1 - p$ with 16 agents and the *Fixed* failure model. Each line and error band represent the mean and 1 standard deviation respectively wrt. 30 runs.

Random failure model. To this end, we implement an environment according to our problem specification and perform a series of MARL experiments using the setup described in Section 5.[1]

[1] The source code is publicly available at https://github.com/hlf-ucl/rlc-2024-rltc.

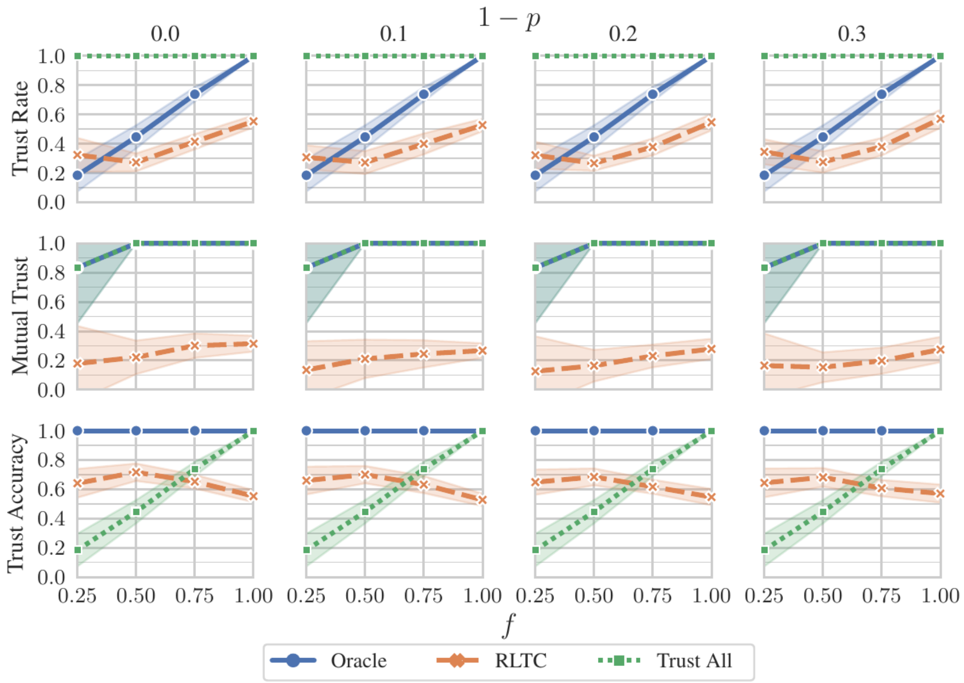
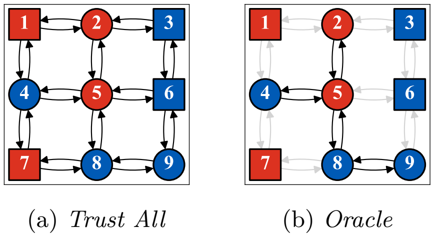


Figure 9: f against trust metrics for fixed $1 - p$ values with 16 agents and *Random* failure model.

6.1 Experimental Setup

We run experiments on both 3×3 and 4×4 square lattices ($N = 9$ and 16 respectively) and both *Fixed* and *Random* failure models. We vary two parameters, namely the fraction f of reliable agents in the system and the value initialization noise $1 - p$. For each failure model, we measure the average consensus success rate and also observe the emergent properties of the learned trust mechanisms. We repeat the experiments 30 times, each one starting with a unique random seed. At the start of each repetition, the reliable and unreliable agents are randomly positioned in the communication graph. Each repetition executes 20,000 training episodes and 2,000 evaluation episodes. Results are averaged over the random seeds.



(a) *Trust All* (b) *Oracle*

Figure 10: Scenarios with baseline agents with fixed trust score arrays (circles) and unreliable agents (squares).

RLTC is compared with two *fixed* baselines: *Trust All* (all agents trust each other) and *Oracle* (each reliable agent knows which of its neighbors are reliable and only trusts them). These baselines are *not* trained using RL and their trust scores are always fixed. We illustrate example trust scores for the baselines in Figure 10. We note that the *Oracle* baseline is provided access to privileged information (i.e., fully accurate knowledge of which neighbors are reliable). Therefore, it is only intended as a comparator, functioning as an upper bound on the success rate that can be achieved by independently trained agents. Please refer to the Appendix for additional experimental details.

Table 1 outlines the metrics that are recorded in our experiments. We assess both the consensus success rate and properties of the trust system that emerges. All metrics are computed exclusively over the subset of *reliable* agents. For the *Trust All* baseline, the average trust rate is always 1 since all agents trust each other. For the *Oracle*, average trust accuracy is 1. This metric describes how far a learned solution differs from the oracle. During the experiments, each metric is computed per timestep, then averaged over all timesteps for an episode-specific statistic.

6.2 Results

Fixed failure model results. We first present the experimental results for 16 agents and the *Fixed* failure model. The observed trends in the 9-agent experiments are similar, thus are moved to the Appendix. Figure 7 (top) and Figure 8 show the effects of varying the fraction of reliable agents on the performance metrics for several fixed values of noise. The values for f and $1-p$ are $\{0.25, 0.5, 0.75, 1.0\}$ and $\{0, 0.1, 0.2, 0.3\}$ respectively.

We observe that fixing the fraction of reliable agents and varying noise has little effect on the output metrics for *RLTC* agents. Increasing $1-p$ only decreases the success rate for *Oracle* and *Trust All* agents, which is expected behavior. For fixed $1-p$, as f increases, the success rate increases as for *Trust All* and *RLTC*; we also note that the success rate for *RLTC* agents is higher than *Trust All*, suggesting that the trust mechanism leads to a statistically significant improvement. In addition, both the average trust rate and mutual trust increase as the reliable fraction increases. However, the values are low compared to the oracle, meaning that the reliable agents do not always trust each other and often prefer their own values v_i over those of their neighbors when performing updates. Low mutual trust also indicates the lack of reciprocation between agents. This is challenging due to our decentralized setup: each agent updates its trust scores locally and independently of others, hence it is difficult to coordinate trust between pairs of agents. The trust accuracy for *RLTC* agents is also relatively low, which means that it deviates from the oracle solution. It shows a decreasing trend as the fraction of reliable agents increases until $f = 0.75$, but increases between $0.75 \leq f \leq 1$ (refer to the Appendix for a further discussion about this phenomenon).

Random failure model results. Similarly, for the *Random* failure model, we present these results for 16 agents in Figure 7 (bottom) and Figure 9. Notice that the success rates are higher in general because the unreliable agents output the correct value 50% of the time. However, *RLTC* exhibits a statistically significant increase in success rate from *Trust All*. This shows that the functionality of our method is not limited to one type of failure model.

Scalability. We also investigate the fundamental dimension of scalability by experimenting with larger communication grid sizes $(5, 6, 7, 8, 9, 10)$ with parameters $f = 0.75$ and $1-p = 0.3$ as an example. The results shown in Figure 11 demonstrate that our decentralized approach maintains similar performance as the number of agents increases.

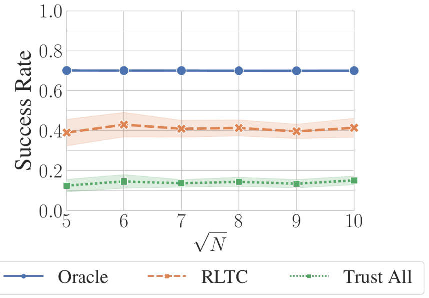


Figure 11: Success rates against the grid dimension $\sqrt{N}$.

7 Conclusion

In this paper, we have investigated the problem of learning consensus in the presence of unreliable agents, a largely neglected yet fundamental issue for the deployment of real-world MARL systems. We have presented *Reinforcement Learning-based Trusted Consensus* (RLTC), a decentralized emergent trust mechanism that allows agents to independently learn which neighbors to trust and which to ignore. Our experiments show that the trust mechanism significantly improves the consensus success rate, indicating that we can deal with unreliable agents effectively while generalizing to different types of failure models and scaling to systems with a greater number of agents. We have also studied properties of the emerging protocol, showing that it leads to low overall trust rate as well as asymmetric trust between agents. Our findings highlight that the proposed trust mechanism enables a decentralized protocol to emerge, which can reduce the impact of unreliable agents in a multi-agent system. RLTC has the potential to be used as a modular component for other, more complex tasks that involve agent coordination.

Acknowledgments

This work was partially supported by The Alan Turing Institute under the UK EPSRC grant EP/N510129/1.

References

Alfarez Abdul-Rahman and Stephen Hailes. A distributed trust model. In *Proceedings of the 1997 Workshop on New Security Paradigms*, pp. 48–60. Association for Computing Machinery, 1998.

Alain Barrat, Marc Barthelemy, and Alessandro Vespignani. *Dynamical Processes on Complex Networks*. Cambridge University Press, 2008.

Jan Blumenkamp and Amanda Prorok. The emergence of adversarial communication in multi-agent reinforcement learning. In *Proceedings of the 5th Conference on Robot Learning (CoRL'21)*, pp. 1394–1414, 2021.

Diane Bouchacourt and Marco Baroni. How agents see things: On visual representations in an emergent language game. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP'18)*, pp. 981–985. Association for Computational Linguistics, 2018.

Francesco Bullo. *Lectures on Network Systems*. Kindle Direct Publishing, 1.6 edition, 2022. URL http://motion.me.ucsb.edu/book-lns.

Christian Cachin, Rachid Guerraoui, and Luis Rodrigues. *Introduction to Reliable and Secure Distributed Programming*. Springer, 2011.

C. Castelfranchi and R. Falcone. Principles of trust for MAS: cognitive anatomy, social importance, and quantification. In *Proceedings of the IEEE International Conference on Multi Agent Systems*, pp. 72–79, 1998.

Filippos Christianos, Lukas Schäfer, and Stefano Albrecht. Shared experience actor-critic for multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

Robin Cohen, Mike Schaekermann, Sihao Liu, and Michael Cormier. Trusted AI and the contribution of trust modeling in multiagent systems. In *Proceedings of the 18th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'19)*, pp. 1644–1648, 2019.

George Coulouris, Jean Dollimore, Tim Kindberg, and Gordon Blair. *Distributed Systems: Concepts and Design*. Pearson Education, 2012.

Allan Dafoe, Edward Hughes, Yoram Bachrach, Tantum Collins, Kevin R McKee, Joel Z Leibo, Kate Larson, and Thore Graepel. Open Problems in Cooperative AI. *arXiv preprint arXiv:2012.08630*, 2020.

Abhishek Das, Théophile Gervet, Joshua Romoff, Dhruv Batra, Devi Parikh, Mike Rabbat, and Joelle Pineau. TarMAC: Targeted multi-agent communication. In *Proceedings of the 36th International Conference on Machine Learning (ICML'19)*, pp. 1538–1546, 2019.

Morris H. DeGroot. Reaching a consensus. *Journal of the American Statistical Association*, 69(345), 1974.

J.A. Fax and R.M. Murray. Information flow and cooperative control of vehicle formations. *IEEE Transactions on Automatic Control*, 49(9), 2004.

Jakob Foerster, Ioannis Alexandros Assael, Nando de Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 29, 2016.

Laura Harding Graesser, Kyunghyun Cho, and Douwe Kiela. Emergent Linguistic Phenomena in Multi-Agent Communication Games. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP'19)*, pp. 3700–3710. Association for Computational Linguistics, 2019.

Mads Haahr. RANDOM.ORG: true random number service. https://www.random.org, 1998–2018. Last Accessed: 17 June 2024.

Aric Hagberg, Pieter Swart, and Daniel S. Chult. Exploring network structure, dynamics, and function using networkx. In *SciPy*, 2008.

Charles R Harris, K Jarrod Millman, Stéfan J van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J Smith, et al. Array programming with NumPy. *Nature*, 585(7825):357–362, 2020.

J. D. Hunter. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 9(3): 90–95, 2007.

Paul L Krapivsky and Sidney Redner. Dynamics of majority rule in two-state interacting spin systems. *Physical Review Letters*, 90(23):238701, 2003.

Leslie Lamport. The part-time parliament. *ACM Transactions on Computer Systems*, 16(2):133–169, 1998.

Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982.

Angeliki Lazaridou, Karl Moritz Hermann, Karl Tuyls, and Stephen Clark. Emergence of Linguistic Communication from Referential Games with Symbolic and Pixel Input. In *Proceedings of the 6th International Conference on Learning Representations (ICLR'18)*, 2018.

Tom Lianza and Chris Snook. A Byzantine failure in the real world - the Cloudflare blog, Nov 2020. URL https://blog.cloudflare.com/a-byzantine-failure-in-the-real-world/. Last Accessed: 17 June 2024.

Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Wes McKinney et al. pandas: a foundational Python library for data analysis and statistics. *Python for High Performance and Scientific Computing*, 14(9):1–9, 2011.

Igor Mordatch and Pieter Abbeel. Emergence of Grounded Compositional Language in Multi-Agent Populations. *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI'17)*, pp. 1495–1502, 2017.

R. Olfati-Saber and J.S. Shamma. Consensus filters for sensor networks and distributed sensor fusion. In *Proceedings of the 44th IEEE Conference on Decision and Control (CDC'05)*, 2005.

Reza Olfati-Saber, J. Alex Fax, and Richard M. Murray. Consensus and cooperation in networked multi-agent systems. *Proceedings of the IEEE*, 95(1):215–233, Jan 2007.

Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *2014 USENIX Annual Technical Conference*, pp. 305–319. USENIX Association, 2014.

Marshall Pease, Robert Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, 1980.

Lerrel Pinto, James Davidson, Rahul Sukthankar, and Abhinav Gupta. Robust adversarial reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning (ICML'17)*, pp. 2817–2826, 2017.

Sarvapali D. Ramchurn, Dong Huynh, and Nicholas R. Jennings. Trust in multi-agent systems. *Knowledge Engineering Review*, 19(1):1–25, 2004.

Murtaza Rangwala and Ryan Williams. Learning multi-agent communication through structured attentive reasoning. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

Wei Ren, R.W. Beard, and E.M. Atkins. A survey of consensus problems in multi-agent coordination. In *Proceedings of the 2005 American Control Conference (ACC'05)*, pp. 1859–1864 vol. 3, 2005.

Jordi Sabater and Carles Sierra. Review on computational trust and reputation models. *Artificial Intelligence Review*, 24(1):33–60, 2005.

Bianca Schroeder and Garth A. Gibson. A large-scale study of failures in high-performance computing systems. *IEEE Transactions on Dependable and Secure Computing*, 7(4):337–350, 2010.

Sainbayar Sukhbaatar, Arthur Szlam, and Rob Fergus. Learning multiagent communication with backpropagation. In *Advances in Neural Information Processing Systems*, volume 29, 2016.

Yanchao Sun, Ruijie Zheng, Parisa Hassanzadeh, Yongyuan Liang, Soheil Feizi, Sumitra Ganesh, and Furong Huang. Certifiably robust policy learning against adversarial multi-agent communication. In *Proceedings of the 11th International Conference on Learning Representations (ICLR'23)*, 2023.

Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the 10th International Conference on Machine Learning (ICML'93)*, pp. 330–337, 1993.

Michael L. Waskom. Seaborn: statistical data visualization. *Journal of Open Source Software*, 6 (60):3021, 2021.

Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, 1992.

Wanqi Xue, Wei Qiu, Bo An, Zinovi Rabinovich, Svetlana Obraztsova, and Chai Kiat Yeo. Misspoke or mis-lead: Achieving robustness in multi-agent communicative reinforcement learning. In *Proceedings of the 21st International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'22)*, 2022.

Huan Zhang, Hongge Chen, Chaowei Xiao, Bo Li, Mingyan Liu, Duane Boning, and Cho-Jui Hsieh. Robust deep reinforcement learning against adversarial perturbations on state observations. In *Advances in Neural Information Processing Systems*, volume 33, 2020a.

Sai Qian Zhang, Qi Zhang, and Jieyu Lin. Succinct and robust multi-agent communication with temporal message control. In *Advances in Neural Information Processing Systems*, volume 33, 2020b.

A Experimental Setting

Implementation. The source code is publicly available at `https://github.com/hlf-ucl/rlc-2024-rltc`. It enables the reproduction of all results presented in the paper, including tables and figures. Our implementation uses public software libraries (Hagberg et al., 2008; Harris et al., 2020; McKinney et al., 2011; Hunter, 2007; Waskom, 2021).

Platform specifications. Experiments are run on Linux (CentOS 7.9.2009) using Python 3.10.3. The platform uses Intel(R) Xeon(R) CPU E5-2637 v4 @ 3.50GHz with 16 cores and 60 GB RAM. Training occurs purely on CPU. The full set of experiments takes approximately 12 hours to run.

Configuration. Table 2 lists the hyperparameters used in all experiments. Multiplicative epsilon decay is used, i.e. $\varepsilon_{t+1} = r^t \varepsilon_0$ at each timestep, where r is a constant decay factor. 20,000 training episodes and 2,000 evaluation episodes are used, each episode with length 30. Results are averaged over 30 random seeds, which were generated using Haahr (1998–2018) in the source code. Table 3

Table 2: RL hyperparameters.

Name	Value
Q-learning step size α	0.03
Discount factor γ	0.999
Initial exploration probability ε_0	0.3
Exploration decay factor r	0.9996

Table 3: Variable ranges.

Name	Values
Number of agents N	9, 16
Fraction of reliable agents f	0.25, 0.5, 0.75, 1.0
Local value initialization noise $1 - p$	0, 0.1, 0.2, 0.3

reiterates the values that were used in experiments from the main paper for convenience (excluding the scalability experiment, which uses more values of N.)

Hyperparameter selection. The RL hyperparameters are selected using a grid search. See Table 4 for the combinations used. We discover that only the discount factor γ and epsilon decay factor r have any noticeable impact on the average reward: using $\gamma = 0.999$ and $r = 0.9996$ converges to the highest value. $\alpha = 0.03$ leads to less noisy rewards over time, and a larger ε_0 allows for more initial exploration.

Table 4: RL hyperparameter grid search.

Name	Values
α	0.03, 0.01, 0.1
γ	0.999, 0.95
ε_0	0.1, 0.3
r	0.9996, 1.0

B Additional Results and Figures

Results for 9 agents, fixed failure model. Figure 12 and Figure 13 show the experimental results for 9 agents (fixed failure model) that were omitted in the main text for brevity, mainly due to similar trends being exhibited.

Plots for 16 agents, fixed failure model, but with constant f per column. Figure 14 shows the same results for 16 agents as seen in the main text, but with $1 - p$ on the x-axes and fixed f per column.

Plots for 16 agents, randomized failure model, but with constant f per column. Figure 15 shows the same results for the random failure model and 16 agents as seen in the main text, but with $1 - p$ on the x-axes and fixed f per column.

Investigating trust accuracies for $0.75 \leq f \leq 1.0$. In the main paper, we notice that the trust accuracy of trained agents decreases when the fractions of reliable agents is between 0 and 0.75, but increases in the 0.75 to 1.0 interval. To investigate this inconsistency, we run finer-grained experiments for values 0.75, 0.8, 0.85, 0.9, 0.95, 1.0. The results are illustrated in Figure 16. We notice a general smooth increase for fixed values of noise, which is an interesting emergent property of the system.

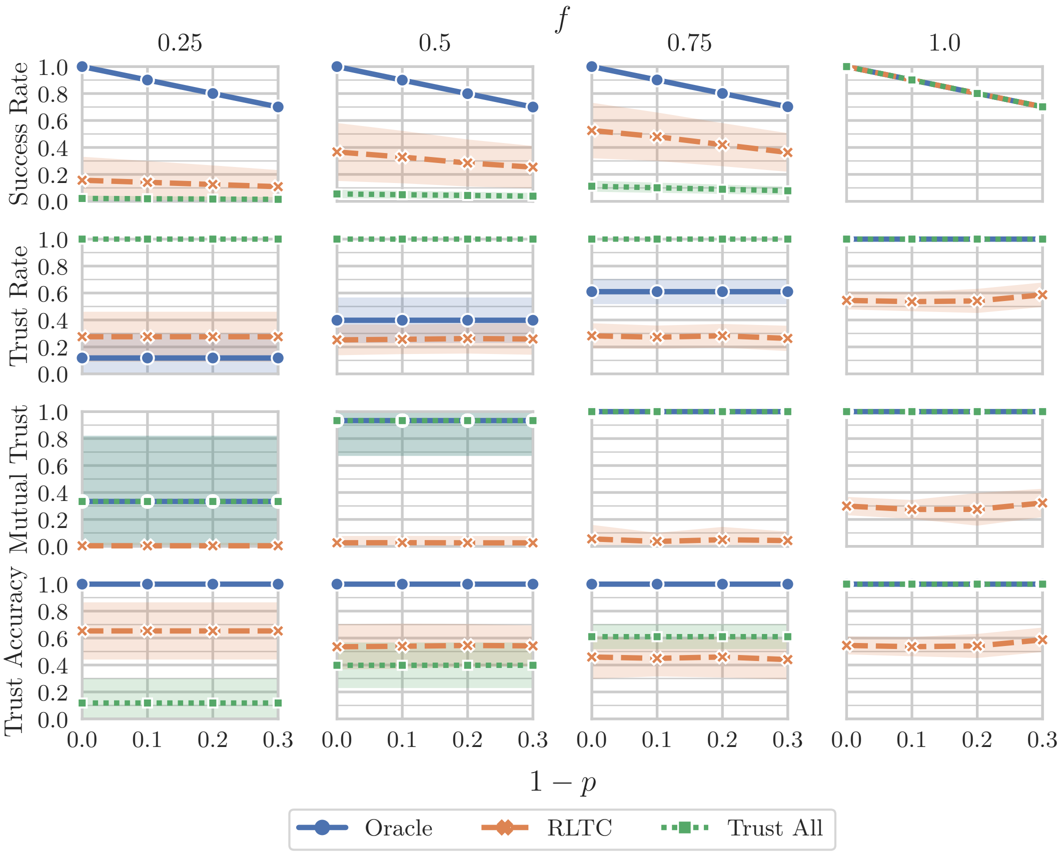


Figure 12: Noise against metric for fixed reliable fractions (9 agents, fixed failure model).

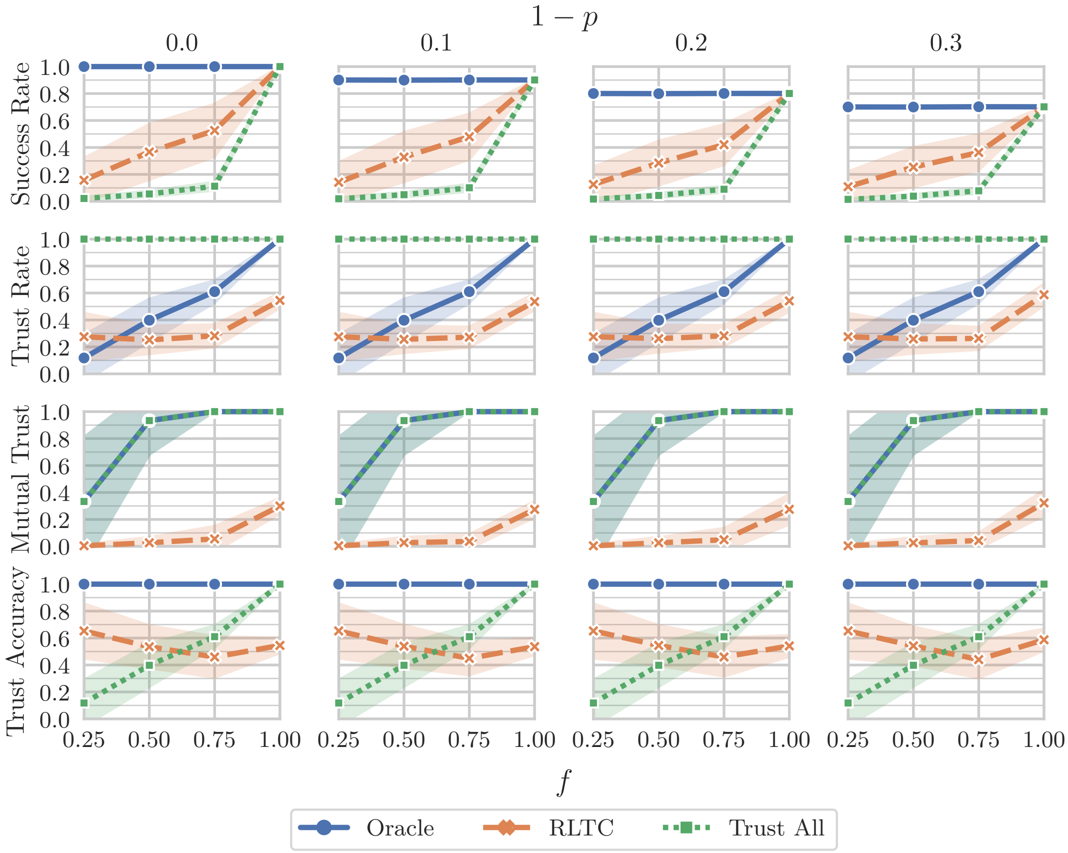


Figure 13: Reliable fraction against metric for fixed noise values (9 agents, fixed failure model).

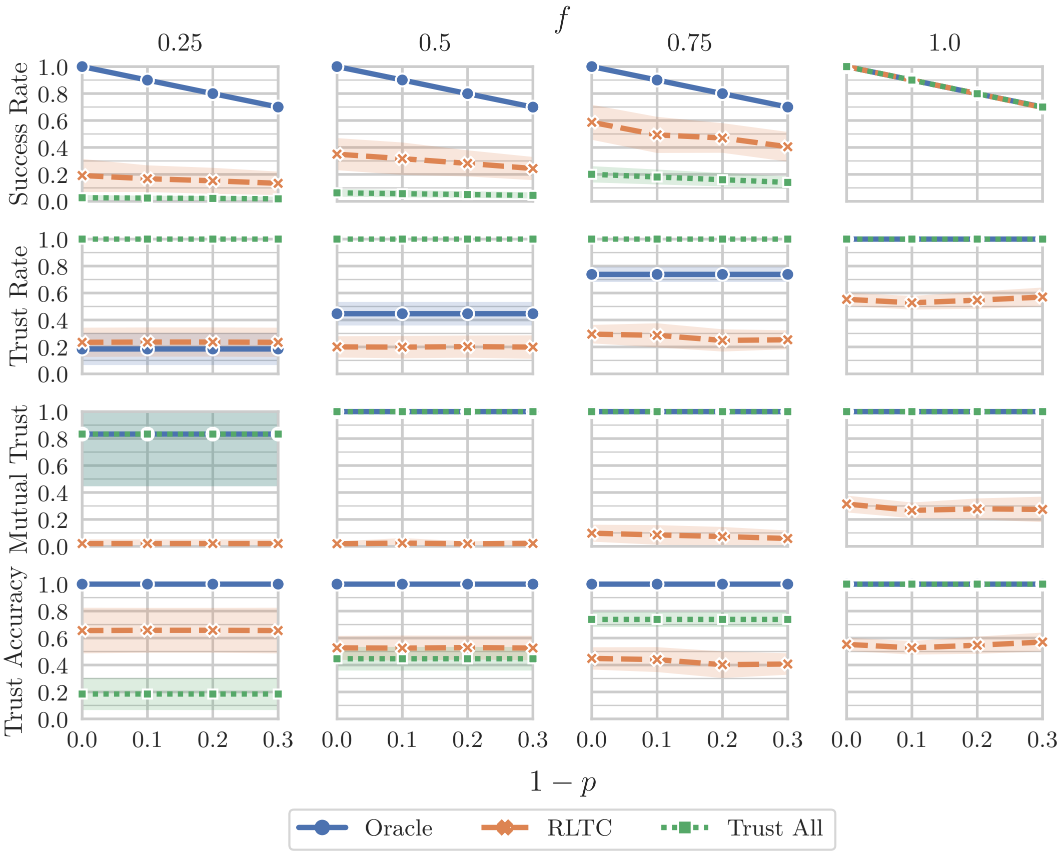


Figure 14: Noise against metric for fixed reliable fractions (16 agents, fixed failure model).

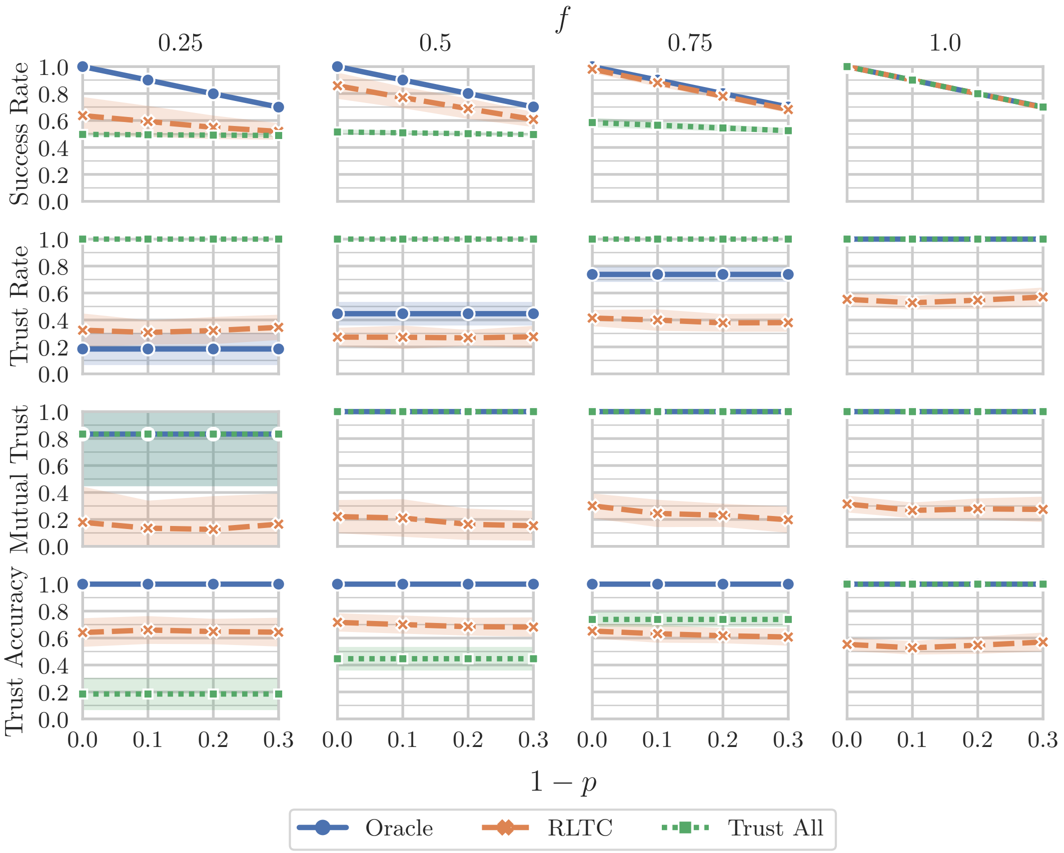


Figure 15: $1 - p$ against metric for fixed f values (16 agents, randomized failure model).

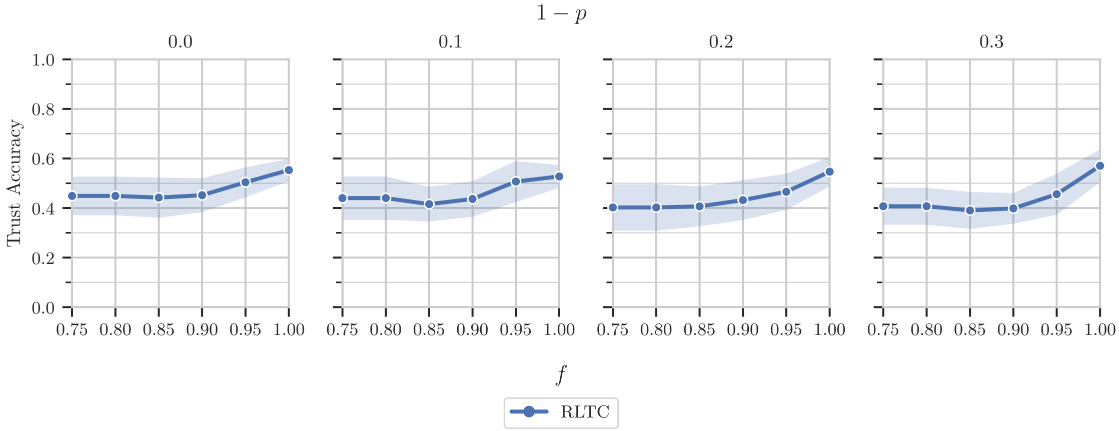


Figure 16: Trust accuracy for $0.75 \leq f \leq 1.0$, 16 agents.