Fault Tolerance in Multi Agent Systems

Savitha Suresh, Akshay Narayan

National University of Singapore
savitha_suresh@u.nus.edu, akshay.narayan@nus.edu.sg

Abstract

Multi-Agent Reinforcement Learning (MARL) has demonstrated strong performance in cooperative and competitive environments, but its deployment in real-world systems remains limited by vulnerability to faulty agents. This work investigates mechanisms for fault tolerance in MARL systems, with a focus on preserving stability, efficiency, and cooperative behavior when subsets of agents fail or behave unpredictably. We propose a framework that integrates behavior masking, reward shaping, and attention mechanisms to mitigate the impact of faulty agents. The masking component enables the policy to selectively downweight agents exhibiting faulty behavior, preventing corrupted trajectories from dominating the learning signal. This is combined with reward shaping strategies, such as penalties for oscillations and inactivity, that guide learning away from failure-prone trajectories. Experimental results in cooperative benchmarks show that our approach significantly improves performance compared to standard MARL baselines. Agents are able to maintain task performance when faced with agent failures. When agents are faulty from the start of an episode, our attention mechanism with behaviour masking achieves a **22% improvement** over the baseline RNN at 20 million steps. In more challenging scenarios where agents become faulty mid-episode, our method achieves a **28% improvement** compared to the baseline, demonstrating stronger robustness under dynamic faults. We further conduct ablation studies to isolate the contribution of behavior masking, showcasing its role in stabilizing training and improving fault tolerance.

Introduction

Reinforcement learning (RL) has emerged as a powerful paradigm for sequential decision-making, where an agent interacts with an environment and learns to maximize cumulative rewards through trial and error. In recent years, the extension of RL to *multi-agent reinforcement learning* (MARL) has attracted increasing attention. In MARL, multiple agents learn simultaneously while interacting with each other and the environment. This setting is particularly relevant for real-world scenarios, where multiple autonomous systems must coordinate or compete to achieve individual or collective goals. The distributed and interactive nature of MARL makes it both powerful and challenging: agents must learn not only how to optimize their own policies, but also how to anticipate, adapt to, and collaborate with other agents.

Despite the progress in MARL, most approaches assume idealized conditions where all agents function correctly and communication channels are reliable. However, real-world systems are far from perfect. Agents may fail due to hardware breakdown, sensor malfunctions, software errors, or adversarial attacks. A single faulty agent may not only degrade its own performance but can also negatively affect the collective behavior of the entire system.

This work explores methods for incorporating fault tolerance into reinforcement learning, with a particular focus on the multi-agent co-ordination. The key idea is that agents should not only maximize rewards under ideal conditions but should also maintain robust performance when some agents are faulty.

The specific research directions pursued in this work include:

- Designing mechanisms for detecting and mitigating faulty agents in MARL environments.
- Developing architectural modifications (e.g., masking, attention mechanisms, transformer-based models) that allow agents to adapt dynamically to faults.
- Incorporating reward shaping strategies that promote resilience, such as penalizing oscillations or rewarding robustness.
- Evaluating the proposed methods on established multi-agent environments, including Resource Gathering (RWARE) and Level-Based Foraging (LBF), with scenarios that simulate agent failures.

Related Work

One of the major challenges in multi-agent systems is ensuring robustness when some agents behave unpredictably or fail. Existing literature has approached this issue from perspectives such as fault-tolerant control, adversarial robustness, and cooperative learning. In this section, we review key works that form the foundation for our study.

Work on fault tolerance in MARL has only recently begun to emerge. A notable example is the paper 'Toward fault tolerance in multi-agent reinforcement learning' (Shi et al. 2025), which introduces a new fault-tolerant MARL

environment based on predator–prey tasks. The authors design four variations: abandonment, where one agent permanently fails; recovery, where surviving agents must retrieve resources from the failed one; navigation, which requires agents to dynamically reschedule tasks in the presence of failures; and patrol, where adversaries can disable agents and the system must adapt. The evaluation shows that conventional MARL algorithms fail to maintain performance under such fault conditions, illustrating the need for approaches explicitly designed with fault tolerance in mind. They employ attention based mechanisms to both the actor and critic and use a priority based replay buffer to aid the agents in tolerating faults.

To tackle mutli agent systems in noisy environments, Kilinc and Montana introduce Multi-agent Deep Deterministic Policy Gradient with a Communication Medium (MADDPG-M) (Kilinc and Montana 2018). This algorithm allows agents to learn to share information through a communication medium, enabling better policy learning. MADDPG-M operates with a two-level, concurrent learning mechanism, where agents determine when to share private observations and learn main task policies, supported by intrinsic rewards. The approach demonstrated performance gains in non-stationary environments compared to established baselines.

Gu et al's paper on, "Attention-Based Fault-Tolerant Approach for Multi-Agent Reinforcement Learning Systems" (Gu, Geng, and Lan 2021), addresses crucial security challenges in multi-agent reinforcement learning (MARL) where agents may exhibit arbitrary faulty or malicious behaviour in harsh environments. Previous state-of-the-art methods, such as MADDPG-M (Kilinc and Montana 2018), were limited by their reliance on prior knowledge of environmental noise intensity, requiring configuration adjustments when noise levels changed. To overcome this, the authors introduce the Attention-based Fault-Tolerant (FT-Attn) model, which employs a multihead attention mechanism to enable agents to selectively identify not only correct but also relevant information from other agents at each time step. This mechanism allows agents to learn effective communication policies alongside their action policies, critically without requiring prior knowledge of the environment's noise intensity. Empirical results demonstrated that FT-Attn consistently surpassed previous methods like MADDPG-M in extremely noisy environments, across both cooperative and competitive scenarios. They analyze cases where a single "true" agent possesses accurate information, and study configurations where this agent is fixed, alternates across episodes, or is positioned in worst-case locations. While the framework contributes insights into robustness against noise and uncertainty, it primarily addresses partial observability rather than explicit agent failures. The connection to fault tolerance is therefore indirect, but the scenarios lay the groundwork for testing how algorithms adapt when reliable information sources become unavailable.

Stanković et al (Stanković and Štula 2013) present a hierarchical multi-agent system designed with fault tolerance in mind. In their framework, worker agents are responsible for task execution, while arbiter agents handle fault detection, mitigation, and recovery. When a worker fails, a distributor delegates fault information to an arbiter, which can recreate failed agents by leveraging stored parent–child task structures. The system ensures that critical information is not lost by replicating it at higher levels of the hierarchy. Experimental results show that the proposed design significantly reduces network and CPU loads under fault conditions compared to baseline solutions. This approach draws heavily from distributed systems principles but demonstrates how hierarchical structures can be adapted for fault-tolerant MARL.

The role of transformers in reinforcement learning has been comprehensively explored by Li et al. in a recent survey (Li et al. 2023). The authors systematically review the motivations and progress behind the growing use of Transformer architectures in Reinforcement Learning (RL), a domain where their evolution has not been well documented. The authors provide a comprehensive overview of Transformer-based RL, offering a taxonomy of existing works and discussing various sub-fields.

Parisotto et al. introduced the Gated Transformer-XL (GTrXL) architecture to address the instability of transformers in reinforcement learning (Parisotto et al. 2020). Unlike conventional Transformer-XL, which often failed to outperform LSTM baselines in RL tasks, GTrXL incorporated gating mechanisms into residual connections and restructured layer normalization to stabilize optimization. Through extensive experiments on DMLab-30 and other partially observable tasks, GTrXL demonstrated superior performance compared to LSTMs and MERLIN, particularly in memory-intensive environments. This work established GTrXL as a stable and expressive sequence model for reinforcement learning, providing a foundation for transformer adoption in both single-agent and multi-agent RL.

To tackle the problem of heterogeneous agent behavior, Wang et al. introduced a personalized behavior-aware framework that integrates both shared knowledge and individual behavior embeddings (Wang et al. 2020). This design allows agents to learn policies that are both collaborative and personalized, ensuring robustness in diverse environments. Their results showed improved adaptability and generalization in cooperative MARL benchmarks. Such personalization aligns with fault tolerance objectives, as it ensures that the learning framework can adapt to agents with distinct or degraded capabilities without collapsing global performance.

Problem Setup

This work was conducted using the EPyMARL library, chosen for its modularity and the availability of boilerplate code, which aided in experimentation across different environments and algorithms. Two environments were chosen for our experiments - RWARE (Multi-Robot Warehouse) and LBF (Level-Based Foraging). The RWARE environment was selected for its close resemblance to real-world warehouse scenarios, featuring delayed rewards and task structures that align with practical robotics applications. In addition, the Level-Based Foraging (LBF) environment was used to explore multi-agent coordination challenges. (Papoudakis

et al. 2021).

Multi-Robot Warehouse The Multi-Robot Warehouse (RWARE) environment is a cooperative, partially observable setting characterized by sparse rewards. It simulates a grid-world warehouse in which multiple robots must locate and transport requested shelves to designated workstations, before returning them to their original positions. Agents receive rewards only upon the successful completion of an entire delivery, which requires executing a precise sequence of actions. Each agent's observation is limited to a local 3×3 grid, containing information about nearby agents and shelves. The observation for each agent comprises a (1,75) tensor: Agent location, carrying status, direction, and 9×7 elements for local surroundings (agents, shelves). The action space A is defined as: {No-op, Turn Left, Turn Right, Forward, Load/Unload Shelf }. Agents receive a reward of 1 for successfully transporting a requested shelf to a designated goal location at the bottom of the warehouse. For the experiments, we use the `rware-tiny-4ag-v2` configuration.

Level Based Foraging In the Level-Based Foraging (LBF) environment (Albrecht and Ramamoorthy 2013), (Albrecht and Stone 2017), agents are required to collect food items scattered randomly across a grid-world. Both agents and items are assigned levels, and an item can be collected if the combined levels of one or more agents meet or exceed the item's level. At each timestep, agents receive observations that either represent the full state of the environment or a restricted view under partial observability. Agents in Level-Based Foraging can choose between six discrete actions at each timestep: {Noop, Move North, Move South, Move West, Move East, Pickup}.

Non-zero rewards are only assigned when food is successfully collected. The reward is determined by both the level of the food item and the levels of the contributing agents. Rewards are normalised such that the total return across all agents in a solved episode equals one, ensuring fair distribution of cooperative outcomes. For the experiments, we use the `Foraging-10x10-4p-4f-v3` configuration.

Fault Models

To simulate faults, one agent is randomly selected with uniform probability. Once selected, the agent is made permanently faulty by forcing it to always take a no-op action. This is achieved by manipulating its action logits and setting an extremely high logit for action 0 and suppressing all others, effectively freezing the agent.

Random Faults By default, faulty agents remain stuck from the beginning of the episode to provide a consistent baseline. To better represent realistic scenarios, we also introduce faults randomly. The chosen agent has a probability `p` of becoming faulty per timestep. Once triggered, the fault is permanent: the agent is forced into a `No-op` state for the remainder of the episode.

Agent Architectures

GRU EPyMARL uses a GRU-based policy architecture by default. Let x_t be the observation at time t. The computation proceeds as follows:

$$\begin{aligned} h_t^{\text{in}} &= \phi(W_1 x_t + b_1) \\ h_t &= \text{GRU}(h_{t-1}, h_t^{\text{in}}) \\ a_t &= W_2 h_t + b_2 \end{aligned}$$

where ϕ denotes a non-linear activation function (e.g., ReLU), and a_t are the raw action logits.

Gated Transformer We followed the gated transformer model as described in literature (Parisotto et al. 2020), utilizing relative multi-head attention and positional encoding. This architecture was chosen for its potential to capture temporal dependencies and model long-range context. Attention heads are expected to focus on past states where agents encountered obstacles or resolved coordination successfully.

Let $x_t \in \mathbb{R}^{d_{\text{in}}}$ denote the observation at time t, and let $X = \{x_{t-\ell+1}, \ldots, x_t\} \in \mathbb{R}^{\ell \times d_{\text{in}}}$ be the input sequence of length ℓ. The Transformer agent processes this sequence as follows:

Input Projection and Normalization. The input sequence is first projected into a higher-dimensional feature space and normalized: $H_0 = \text{LayerNorm}(\phi(W_1 X + b_1))$, where $W_1 \in \mathbb{R}^{d_{\text{model}} \times d_{\text{in}}}$, $b_1 \in \mathbb{R}^{d_{\text{model}}}$, and ϕ denotes a non-linear activation function (e.g., ReLU). The result is $H_0 \in \mathbb{R}^{\ell \times d_{\text{model}}}$.

Stacked Decoder Blocks with Gated Residuals. The model consists of L decoder blocks. Each block integrates contextual information using a gated residual connection over a relative self-attention mechanism. For each layer $l \in \{1, \ldots, L\}$:

$$\tilde{H}_l = \text{GRU}_1^{(l)}\left(H_{l-1}, \text{SelfAttn}^{(l)}\left(\text{LN}(H_{l-1}), K_l, V_l\right)\right),$$

$$H_l = \text{GRU}_2^{(l)}\left(\tilde{H}_l, \text{FFN}^{(l)}(\text{LN}(\tilde{H}_l))\right),$$

where:

- $K_l, V_l \in \mathbb{R}^{(m+\ell) \times d_{\text{model}}}$ are key/value sequences formed by concatenating memory and current input,
- $\text{GRU}_1^{(l)}$, $\text{GRU}_2^{(l)}$ are GRU-inspired gating modules,
- $\text{FFN}^{(l)}$ is a feedforward network (typically a linear layer followed by ReLU),
- *LN* is LayerNorm and $SelfAttn$ is Self Attention.

Output Projection. Finally, the output of the top decoder block is normalized and projected to raw action logits:

$$\begin{aligned} H_{\text{out}} &= \text{LayerNorm}(H_L), \\ a_t &= W_2 H_{\text{out}} + b_2, \end{aligned}$$

= where $W_2 \in \mathbb{R}^{d_{\text{out}} \times d_{\text{model}}}$, $b_2 \in \mathbb{R}^{d_{\text{out}}}$, and $a_t \in \mathbb{R}^{\ell \times d_{\text{out}}}$ are the output logits over actions at each timestep.

Training Algorithm

All agents were trained using the MAPPO (Multi-Agent Proximal Policy Optimization) algorithm as implemented in EPyMARL.

Multi-Agent Proximal Policy Optimization (MAPPO). Proximal Policy Optimization (PPO) is a widely used on-policy policy gradient method that updates the policy by maximizing a clipped surrogate objective, which ensures stable learning by preventing large policy updates. In the multi-agent setting, Multi-Agent PPO (MAPPO) extends PPO to cooperative or mixed cooperative-competitive environments with multiple agents. Each agent i maintains its own policy $\pi_{\theta_i}(a_i|o_i)$, where o_i is the local observation and a_i is the action. A centralized critic is often employed during training, which estimates the value function $V(s)$ or the action-value function $Q(s,a)$ using the global state s and potentially the joint action $a = (a_1, \ldots, a_n)$. MAPPO has shown strong empirical performance in a range of benchmark multi-agent environments by stabilizing training while maintaining sample efficiency.

Fault Tolerance Aiding Methods

Reward Shaping To discourage agents from becoming stuck or oscillating between positions, we introduced two types of reward penalties:

Stuck Penalty: Applies an exponential penalty that increases the longer an agent remains stationary.

Oscillation Penalty: Penalizes agents revisiting the same set of positions in quick succession, indicating back-and-forth movement.

These penalties are computed post-environment step and subtracted from the environment-provided rewards.

Proposed Method

We propose a transformer-based multi-agent architecture in the policy network that integrates self-attention, behavior-conditioned cross-attention, and gating to effectively model temporal and inter-agent dynamics. (GrtXL-BCA)

Overview Let $X \in \mathbb{R}^{B\cdot N\times T\times d_{\text{in}}}$ denote the input observations, where B is the batch size, N the number of agents, T the sequence length, and d_{in} the input dimension. Each agent's trajectory is processed through the following key components:

1. Self-Attention with Relative Positional Encoding
2. GRU-inspired Gating Mechanism
3. Cross-Attention Conditioned on Behavioral Similarity
4. Feedforward Network

Self-Attention with Relative Positional Encoding Each agent processes its own temporal sequence via a relative multi-head attention mechanism. For input sequence $X \in \mathbb{R}^{T\times d}$ (per agent), the model first derives the standard query, key, and value representations. The attention score between timestep t and t' is computed as:

$$A_{tt'} = \frac{1}{\sqrt{d}}\left[(Q_t + u)^\top K_{t'} + (Q_t + v)^\top R_{t-t'}\right] \tag{1}$$

where $u, v \in \mathbb{R}^d$ are global learnable vectors for content and position-based attention, respectively. The attention weights $\alpha_{tt'}$ are computed by applying a softmax to the attention scores $A_{tt'}$ across all timesteps. The output of the self-attention layer at timestep t, denoted as $\text{SelfAttn}(X)_t$, is then obtained as a weighted sum of the value vectors $V_{t'}$, using these attention weights.

This output is gated using a GRU-like gating unit:

$$H^{(1)} = \text{GRUGate}(X, \text{SelfAttn}(X)) \tag{2}$$

Cross-Agent Attention via Behavior Clustering To enable inter-agent coordination, we introduce a cross-attention mechanism where each agent attends to behaviorally similar agents. Since faulty agents are anomalous, attending to agents which behave similarly will stabilize learning and help in achieveing good perfomance. To enable the policy to adapt dynamically, each agent receives an input sequence that includes observations of other agents, interleaved such that more recent timesteps appear later in the sequence. The resulting input tensor for a single agent is denoted as $X \in \mathbb{R}^{(N\cdot T)\times d}$, where N is the number of agents, T the temporal horizon, and d the feature dimension.

First, a similarity network $\phi(\cdot)$ encodes each agent's current observation x_i into an embedding $e_i \in \mathbb{R}^{d'}$:

$$e_i = \phi(x_i), \quad \hat{e}_i = \frac{e_i}{\|e_i\|_2} \tag{3}$$

The pairwise cosine similarity matrix $S \in \mathbb{R}^{N\times N}$ is computed:

$$S_{ij} = \hat{e}_i^\top \hat{e}_j \tag{4}$$

A binary attention mask $M \in \{0,1\}^{N\times N}$ is derived:

$$M_{ij} = \mathbb{I}(S_{ij} > \tau) \tag{5}$$

We denote each agent's temporally encoded representation as $H_i \in \mathbb{R}^{T\times d}$. The standard query ($Q$), key ($K$), and value ($V$) matrices are calculated using the input sequence and temporal representation. For agent i, cross-attention is computed over other agents j where $M_{ij} = 1$:

$$\alpha_{i,t,t'} = \frac{\exp(Q_{i,t}^\top K_{j,t'})}{\sum_{j',t''}\exp(Q_{i,t}^\top K_{j',t''})} \cdot M_{ij} \tag{6}$$

The cross-attention output, denoted as $\text{CrossAttn}_i(t)$, is obtained as a weighted sum of the value representations $V_{j,t'}$, where the attention score $\alpha_{i,t,t'}$ determines the contribution of each agent j at timestep t'.

This output is again gated using a GRU:

$$H_i^{(2)} = \text{GRUGate}(H_i^{(1)}, \text{CrossAttn}_i) \tag{7}$$

Feedforward and Output Projection The agent representation is refined with a feedforward network:

$$H_i^{(3)} = \text{GRUGate}(H_i^{(2)}, \text{ReLU}(H_i^{(2)}W_1 + b_1)W_2 + b_2) \tag{8}$$

The final action logits for agent i are computed by applying a linear transformation to its final hidden representation $H_i^{(3)}$.

Memory Mechanism A limited-length memory bank is maintained per layer, allowing temporal context to persist across episodes. After each forward pass, hidden states are appended to the memory, truncated to a fixed size L_{mem}.

Auxiliary Loss for Clustering If ground-truth agent cluster labels y_i are available, the similarity network is trained

with an auxiliary contrastive loss:

$$\mathscr{L} = \sum_{i<j} [\mathbb{I}(y_i = y_j)(1 - S_{ij}) + \mathbb{I}(y_i \neq y_j) \cdot \text{ReLU}(S_{ij} - m)] \tag{9}$$

where m is a margin hyperparameter.

Experiments

All the experiments were conducted in H100 or A100 GPUs. All results represent the values during evaluation except the figures representing the convergence of the networks.

No Faults Both RNN and transformer were run with 4 agents and for 40M timesteps without any faults to establish a baseline in RWARE. In LBF the same experiments were done for 20M timesteps. Table 1 illustrates the reward obtained from the different architectures. We used simple architecture for the transformers with 1 layer and 2 heads for the experiments.

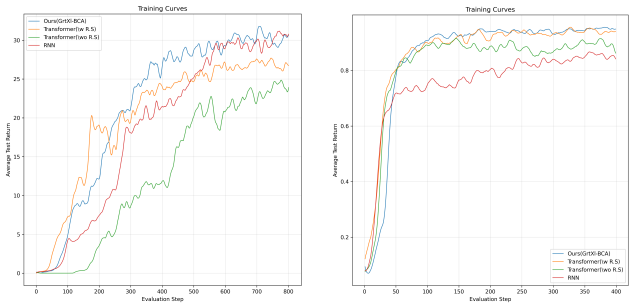
Environment	Architecture	Reward
rware-tiny-4ag	RNN	47.8
	Transformer	**49.4**
lbf4p-4f	RNN	0.94
	Transformer	**0.95**

Table 1: Total reward for different architectures in RWARE and LBF environments.

Training with a single agent being faulty The faults are induced during training to understand how the existing model learns to tolerate faults without any changes.

The raw Transformer underperforms relative to the RNN baseline. While it converges quickly, it plateaus at a lower reward, indicating poorer overall performance. In contrast, our proposed solution not only achieves higher rewards early in training, but also converges to a higher final reward than RNN. This demonstrates improved learning efficiency. Figure: 1 represents the evaluation reward curves for the various architectures. Table: 2 gives the reward of the different architectures in RWARE and LBF. Rware environments are trained for 40M and LBF environments are trained for 20M timesteps.

Random faults As mentioned in the previous sections, faults are not consistent and a healthy agent might become faulty randomly in the middle of an episode, hence we conducted experiments to measure the performance of the proposed solution in such scenarios which simulate the reality better. Since the rewards for each run has high variance for RNN, the mean rewards with the standard deviation of RNN and Transformer architectures with the different modifications, obtained during evaluation, has been reported in the Table: 3 and the maximum rewards obtained are reported in Table: 4. From the results, it is clear that our proposed solution is stable, consistent and produces much higher rewards than RNN. Though the highest reward produced by the RNN architecture is 25, it is rare and the training often collapses



(a) RWARE (b) LBF

Figure 1: RNN vs Transformer evaluation reward curves for RWARE and LBF without agent ID in the observations.

Env	Arch	0	1	2	3
RWARE	RNN	28.56	28.20	28.50	28.52
	GrtXl (\wo R.S)	25.90	25.90	26.25	25.75
	GrtXl (\w R.S)	27.60	27.11	27.19	26.84
	Ours	**31.43**	**32.19**	**32.30**	**31.78**
LBF	RNN	0.81	0.84	0.76	0.56
	GrtXl (\wo R.S)	0.90	0.73	0.62	0.62
	GrtXl (\w R.S)	0.98	0.82	0.67	0.72
	Ours	**0.72**	**0.67**	**0.82**	**0.97**

Table 2: Rewards for each agent being faulty in RWARE and LBF environments. GrtXl = Gated Transformer, R.S = Reward Shaping, Ours = Gated Transformer with Behavioural Cross Attention. Agent ids are not encoded in the observation.

to 0. But our solution is stable and the training does not collapse. Figure: 2 reports the average evaluation reward curves for 5 different seeds for RWARE and LBF. The solid line represents the mean curve, while the shaded region around it indicates the variance.

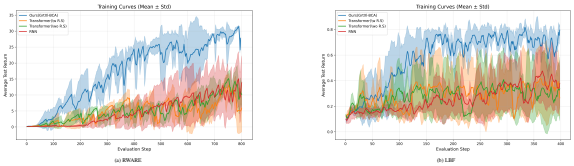


Figure 2: Average evaluation reward curves of models when agents become faulty randomly mid-episode over 5 different seeds. RNN = Baseline, R.S = Reward Shaping, GrtXl-BCA = Gated Transformer with Behavioural Cross Attention

Multiple Faulty Agents To evaluate the consistency of the proposed solution, we tested scenarios with multiple faulty agents. Only the random fault scenario is reported, as it is more realistic and effectively reflects the stability of the models. The results are summarized in Table 5.

Overall, the results demonstrate that the proposed solution is stable, consistently achieving high rewards across multiple runs, and generalizes well across fault scenarios. Un-

Environment	Architecture	0	1	2	3
RWARE	RNN	16.40	16.69	16.36	16.59
	Transformer (\wo R.S)	17.83	17.54	17.90	17.48
	Transformer (\w R.S)	16.75	16.24	16.70	16.90
	Ours (GrtXl-BCA)	**30.70**	**30.56**	**30.64**	**30.58**
LBF	RNN	0.50	0.51	0.51	0.49
	Transformer (\wo R.S)	0.41	0.43	0.43	0.41
	Transformer (\w R.S)	0.44	0.43	0.40	0.41
	Ours (GrtXl-BCA)	**0.83**	**0.81**	**0.78**	**0.78**

Table 3: Average evaluation results for each agent ID being faulty randomly mid-episode in RWARE and LBF environments over 5 different seeds. Transformer = Gated Transformer, R.S = Reward Shaping, CA = Cross Attention, Ours = Gated Transformer with Behavioural Cross Attention.

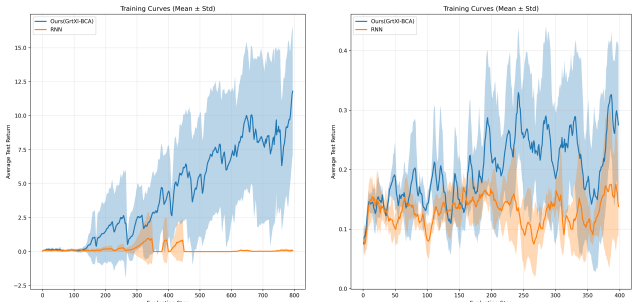
Environment	Architecture	0	1	2	3
RWARE	RNN	25.07	25.00	24.93	26.02
	Transformer (\wo R.S)	20.82	20.70	21.25	20.45
	Transformer (\w R.S)	19.74	19.79	20.54	19.73
	Ours (GrtXl-BCA)	**32.21**	**31.49**	**31.80**	**31.89**
LBF	RNN	0.73	0.76	0.77	0.70
	Transformer (\wo R.S)	0.77	0.80	0.80	0.74
	Transformer (\w R.S)	0.82	0.80	0.72	0.73
	Ours (GrtXl-BCA)	**0.89**	**0.84**	**0.80**	**0.81**

Table 4: Maximum rewards during evaluation for each agent ID being faulty randomly mid-episode in RWARE and LBF environments over 5 different seeds. Transformer = Gated Transformer, R.S = Reward Shaping, CA = Cross Attention, Ours = Gated Transformer with Behavioural Cross Attention.

like the RNN baseline, which often collapses during training or produces highly variable performance, our method maintains steady learning and performs reliably even when agents are randomly or strategically faulty. This highlights the robustness and suitability of our model. Figure: 3 denotes the average reward curves of RNN and Gated Transformer with Behavioural Cross Attention when two agents are randomly faulty over 5 different seeds. Other variants of the Transformer have been not been reported because the previous experiments have already proved that our model outperforms them.

Environment	Architecture	Rewards
RWARE	RNN	1.26
	Ours (GrtXl-BCA)	**10.25**
LBF	RNN	0.26
	Ours (GrtXl-BCA)	**0.411**

Table 5: Average Evaluation results for two agents being randomly faulty in RWARE and LBF environments over 5 different seeds. Ours = Gated Transformer with Behavioural Cross Attention.



(a) RWARE (b) LBF

Figure 3: Average evaluation reward curves of models when two agents become faulty randomly mid-episode over 5 different seeds. RNN = Baseline, R.S = Reward Shaping, GrtXl-BCA = Gated Transformer with Behavioural Cross Attention

Conlucsion and Future Work

This work investigated the integration of fault tolerance into reinforcement learning (RL) and, in particular, multi-agent reinforcement learning (MARL). Modern RL has achieved significant success in diverse domains ranging from Atari games to robotic control. However, the majority of existing frameworks assume idealized settings where agents and their interactions are perfectly reliable. In contrast, real-world systems, such as autonomous driving fleets, surgical and distributed robotics, must contend with the possibility of agent failures, noisy communication, and degraded performance. Addressing these issues requires explicitly embedding fault tolerance into the design of learning algorithms.

The primary contribution of this work was to explore methods that allow multi-agent systems to remain robust in the presence of faulty agents. To this end, we examined strategies such as behaviour masking mechanisms, which can filter out unreliable features or agents, and attention-based architectures such as Gated Transformer-XL (GTrXL), which offer stability and long-range dependency modeling. By combining these architectural innovations with policy gradient methods like Multi-Agent PPO (MAPPO), we designed a framework where agents can adaptively ignore or compensate for failures in their peers, thereby improving the system's overall resilience.

Through empirical studies on benchmark environments such as the Multi-Robot Warehouse (RWARE) and Level-Based Foraging (LBF), this work demonstrated that incorporating fault tolerance mechanisms can mitigate the negative impact of failed agents. In scenarios where one or more agents were constrained to perform no actions - simulating hardware failures, the remaining agents were able to adapt and continue fulfilling collective objectives. These findings underline the importance of embedding fault-awareness into MARL and provide a pathway toward safer and more reliable deployment of such systems in practical applications.

Limitations Despite these promising results, several limitations remain. The environments RWARE and LBF, while widely used benchmarks, are still relatively simple com-

pared to real-world applications. The notion of "fault" in these experiments was modeled primarily as no-op behavior, whereas in practice, faults may take on more complex forms such as adversarial behavior, stochastic dropouts, or partial communication failures. Extending fault tolerance mechanisms to these more challenging conditions is a natural next step.

Another limitation is in the computational demands of transformer-based architectures. Although GTrXL stabilized training, it also introduced higher model complexity and resource requirements compared to recurrent baselines. This constraint may hinder scalability in large multi-agent systems or in real-time applications where inference speed is critical. Balancing model expressiveness with computational efficiency will therefore be essential for future progress.

Third is the stabilization of the critic network - all the architecture changes were done only to the policy network. As the faults become more random, the training gets destabilized and collapses. To prevent this the critic has to incorporate techniques to handle the distribution shfit.

Future Work Future work entails exploration of richer fault models, more adaptible behavioural masking techniques and scale. Beyond no-op behavior, agents could be modeled as intermittently failing, exhibiting stochastic noise, or even acting adversarially. Future work should also explore scaling these methods to more complex, high-dimensional environments. Combining visual perception (e.g., with vision transformers) with fault-tolerant MARL would enable applications in robotics, autonomous driving, and other safety-critical fields. Designing learning systems that can adapt to these more challenging areas will bring MARL closer to deployment in high-stakes domains such as defense, healthcare, or financial systems.

In summary, this work has argued that fault tolerance is not a peripheral concern but a core requirement for the deployment of reinforcement learning in real-world, safety-critical domains. By developing and testing mechanisms such as masking strategies and transformer-based architectures, we have shown that agents can remain effective even in the presence of faults. While challenges remain, the trajectory of this research suggests a future where multi-agent reinforcement learning systems can be both powerful and resilient. Achieving this balance will be crucial for the next generation of artificial intelligence, where reliability and robustness are as important as raw performance.

References

Albrecht, S. V.; and Ramamoorthy, S. 2013. A Game-Theoretic Model and Best-Response Learning Method for Ad Hoc Coordination in Multiagent Systems. In *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 1155–1156. Saint Paul, Minnesota, USA: International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS).

Albrecht, S. V.; and Stone, P. 2017. Reasoning about Hypothetical Agent Behaviours and their Parameters. In *Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 547–556. São Paulo, Brazil: International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS).

Gu, S.; Geng, M.; and Lan, L. 2021. Attention-Based Fault-Tolerant Approach for Multi-Agent Reinforcement Learning Systems. *Entropy*, 23(9): 1133.

Kilinc, O.; and Montana, G. 2018. Multi-agent Deep Reinforcement Learning with Extremely Noisy Observations. Technical Report arXiv:1812.00922, arXiv.

Li, W.; Luo, H.; Lin, Z.; Zhang, C.; Lu, Z.; and Ye, D. 2023. A Survey on Transformers in Reinforcement Learning. *Transactions on Machine Learning Research*, 2023. Publisher Copyright: © 2023, Transactions on Machine Learning Research. All rights reserved.

Papoudakis, G.; Christianos, F.; Schäfer, L.; and Albrecht, S. V. 2021. Benchmarking Multi-Agent Deep Reinforcement Learning Algorithms in Cooperative Tasks. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*. 35th Conference on Neural Information Processing Systems (NeurIPS) — Datasets and Benchmarks Track.

Parisotto, E.; Song, F.; Rae, J.; Pascanu, R.; Gulcehre, C.; Jayakumar, S.; Jaderberg, M.; Kaufman, R. L.; Clark, A.; Noury, S.; Botvinick, M.; Heess, N.; and Hadsell, R. 2020. Stabilizing Transformers for Reinforcement Learning. In III, H. D.; and Singh, A., eds., *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, 7487–7498. PMLR.

Shi, Y.; Pei, H.; Feng, L.; Zhang, Y.; and Yao, D. 2025. Toward Fault Tolerance in Multi-Agent Reinforcement Learning. *IEEE Transactions on Automation Science and Engineering*, 22: 19007–19024.

Stanković, R.; and Štula, M. 2013. Fault Tolerance through Interaction and Mutual Cooperation in Hierarchical Multi-Agent Systems. In *Proceedings of the 5th International Conference on Agents and Artificial Intelligence (ICAART 2013)*. SCITEPRESS – Science and Technology Publications, Lda.

Wang, T.; Dong, H.; Lesser, V.; and Zhang, C. 2020. ROMA: multi-agent reinforcement learning with emergent roles. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org.