Large Language Models are Autonomous Cyber Defenders

Sebastián R. Castro, Roberto Campbell, Nancy Lau, Octavio Villalobos, Jiaqi Duan, Alvaro A. Cardenas
University of California, Santa Cruz

***Abstract*—Fast and effective incident response is essential to prevent adversarial cyberattacks. Autonomous Cyber Defense (ACD) aims to automate incident response through Artificial Intelligence (AI) agents that plan and execute actions. Most ACD approaches focus on single-agent scenarios and leverage Reinforcement Learning (RL). However, ACD RL-trained agents depend on costly training, and their reasoning is not always explainable or transferable. Large Language Models (LLMs) can address these concerns by providing explainable actions in general security contexts. Researchers have explored LLM agents for ACD but have not evaluated them on multi-agent scenarios or interacting with other ACD agents. In this paper, we show the first study on how LLMs perform in multi-agent ACD environments by proposing a new integration to the CybORG CAGE 4 environment. We examine how ACD teams of LLM and RL agents can interact by proposing a novel communication protocol. Our results highlight the strengths and weaknesses of LLMs and RL and help us identify promising research directions to create, train, and deploy future teams of ACD agents.**

***Index Terms*—Autonomous Cyber Defense, Incident Response, Large Language Models, Reinforcement Learning, AI Agents.**

I. Introduction

Cyber threats are evolving rapidly, becoming more complex, frequent, and costly for organizations. A report by Checkpoint shows that attacks are at all-time highs, with average weekly cyberattacks per organization at 1800 compared to 800 in Q3 of 2024 [1].

Traditionally, responding to intrusion alerts requires expert human operators who analyze incidents and recover from attacks. As the sophistication of cyberattacks increases [2], manually managing security responses becomes increasingly challenging. To address this issue, automation is needed not only to detect but also to respond to attacks in real-time. Autonomous Cyber Defense (ACD) approaches this using AI agents to detect and mitigate attacks.

ACD agents have been primarily based on Reinforcement Learning (RL). Still, their applications to real-world systems are constrained by (1) limited explainability, (2) limited transferability to other environments (different attackers or different networks), and (3) training challenges due to complex realistic environment creation and RL sample inefficiency [3].

To address the limitations of RL-based approaches, we study whether advances in LLMs can disrupt them. First, LLMs can provide explanations for their decisions. Second, they have been trained with data from diverse threat models and networks. Third, we can use pre-trained models (without the need to develop a gym environment).

Before LLM agents can be used for ACD, we need to answer several research questions including; How can we train, design, and deploy teams of ACD agents? How do LLMs perform as ACD agents compared to RL agents? Can LLMs address the limitations of RL for ACD by providing a practical, human-interpretable reasoning of agent strategy? Can LLM ACD agents make sound decisions and generalize against different adversaries? What challenges must LLMs address towards ACD?

To answer these questions, we propose the following contributions:

1) **LLM+RL ACD Framework**: We develop the first study on how LLMs perform in multi-agent ACD environments by creating a framework to integrate LLMs into CybORG [4], the most prominent simulation environment for ACD and RL. We make our framework open source [1].
2) **ACD Multi-agent Communication Protocol**: We introduce and evaluate the first communication protocol for diverse ACD agents (RL and LLM) in our experiments.
3) **Evaluation**: We evaluate RL and LLM agents against a diverse adversary set, discuss LLM advantages over RL to improve ACD agents. We identify promising directions for future research, including model and prompt tuning, and goal-based planning.

II. Background

ACD focuses on creating AI agents that can respond to incidents fast and accurately [5], [6]. Typically, these agents are trained in simulated environments called ACD gyms. These gyms provide a testbed that recreates a simulated network where agents interact with the environment and learn to defend against attackers [7]. Well-designed gyms offer a variety of scenarios to effectively explore state spaces [8], ensure robustness, and simulate realistic scenarios [7], [9]. Examples of gyms include CyberBattleSim [10], NASimEmu [11], and FARLAND [12]. However, these frameworks are not all publicly accessible, actively maintained, or are limited to to particular adversarial scenarios.

One of the most relevant ACD environments is CybORG [4], the foundation for the CAGE Challenges [13]. CybORG offers a high-level abstraction for diverse adversary emulation scenarios (e.g., drone and enterprise networks). The CAGE Challenge is an annual Technical Cooperation Program

Proceedings to appear: IEEE CAI 2025 Adaptive CyberDefense Workshop.

[1] https://github.com/r4wd3r/llms-are-acd

(TTCP) competition to advance the state-of-the-art ACD. Using the CybORG gym, participants in this challenge submit defender agents (blue agents) to remove attackers (red agents) and maximize availability for network users while protecting critical services.

CAGE challenges evaluate community submissions based on the highest joint reward obtained in the simulations. Previous submissions to CAGE challenges use RL agents. Its latest edition, the CAGE 4 Challenge, includes a multi-agent environment with limited communication between defensive agents, partial network observability, and a shared reward between agents. In a submission for this challenge, Singh et al. [14] proposed a hierarchical RL approach by decomposing the action space into a meta-policy and three sub-policies. Their results show that a strong expert meta-policy can outperform a joint Proximal Policy Optimization (PPO) policy [15]. In another submission, the Cybermonic team proposed KEEP, a variant of PPO with Graph Convolutional Networks (GCN) [16]. KEEP structures observations as heterogeneous graphs of entities (e.g., files, hosts, routers, and ports) where agent actions alter these graphs. At the time of writing, Cybermonic's KEEP is the only open-source solution for the CAGE 4 competition's leaderboard. So, we use it as a baseline for our evaluations.

Recent work has also started exploring LLM agents for ACD. Rigaki et al. [17] propose an LLM-based ReACT attacker for the NetSecGame environment, and Yan et al. [18] proposed a closed-source trained RL agent that provides action feedback to an LLM implemented for the CAGE 1 challenge. Yet these approaches have only been implemented in single-agent simulations and have not explored LLM decision making compared with RL. In this paper, we introduce the first integration of both LLM and RL agents on the multi-agent CAGE 4 challenge and propose approaches to analyze LLM reasoning for action selection. We think these steps will take us closer to understanding how generative AI can help us design future ACD agents for realistic incident response.

A. CybORG CAGE 4

The CAGE 4 Challenge simulates a Multi-Agent Reinforcement Learning (MARL) scenario with a team of defender agents (*blue agents*) protecting a network from adversaries (*red agents*) while maintaining service availability for users (*green agents*). As seen in Fig. 1, the network is divided into zones that are protected by the *blue agents* as follows: two deployed networks (A and B), each containing restricted and operational zones; a headquarters network with public access, admin and office zones; and a contractor network where the *red agent* starts. *Green agents* can use all the systems in the network, but they have a slight probability of giving the *red agent* access to a random host by falling victim to a "phishing attack". Each agent has a set of actions defined in Table I.

CAGE 4 progresses through 3 phases (Planning, Mission A, Mission B), each with different communication constraints between network zones. Specific operational zones become isolated in active missions, while restricted zones have limited connectivity. Blue agents must adapt their strategies to changing policies, such as blocking unauthorized connections while ensuring that critical services remain available to legitimate users (green agents).

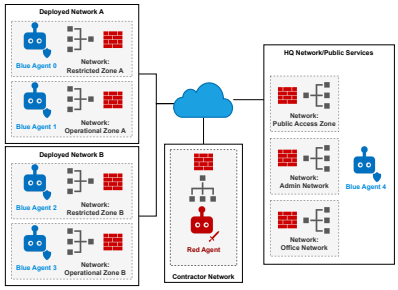

Fig. 1: CAGE 4 Challenge Architecture

Blue	Red	Green
Monitor	*Discover*	*LocalWork*
Analyse	*Exploit*	*AccessService*
DeployDecoy	*PrivilegeEscalate*	
Remove	*DegradeService*	
Restore	*Impact*	
AllowTrafficZone/BlockTrafficZone	*Withdraw*	

TABLE I: List of actions per agent

The reward function for *blue agents* is designed to receive penalties if *green agents* cannot use the resources. This can happen due to defensive actions such as blocking a network or restoring a host (which takes the host out of operation for a while) or when *red agents* impact their systems.

III. LLM Agents for Autonomous Cyber Defense

We created an LLM-based agent for CybORG CAGE 4 [4]. To our knowledge, this is the first multi-agent implementation of an LLM interacting with other ACD agent types, including RL ones. In this section, we discuss the challenges and our design.

A. LLM Adapter Framework

To understand how LLMs perform as ACD agents in simulated/emulated scenarios, we created an extensible adapter framework to integrate LLMs into CybORG. Its development imposes challenges in terms of performance and cost optimization. During its test phase, CybORG is designed to run with offline post-training RL policies. Thus, the action selection is notably faster than LLMs inference time. In our experiments, LLM inference for action selection with an online small model (i.e., *GPT-4o-mini* [19]) was in average **150 times slower** than action selection through a post-training RL policy. To overcome this, when developing initial versions of our adapter, we relied on light offline models (e.g., *TinyLLaMA*) and GPU optimizations, removing costs for paid models and added overhead due to network latency and inference time.

Fig. 2 shows our LLM adapter and its workflow for action selection using these models. We highlight our LLM adapter and how it interacts with the agents when the CAGE 4 simulation starts. The figure shows the scenario where one

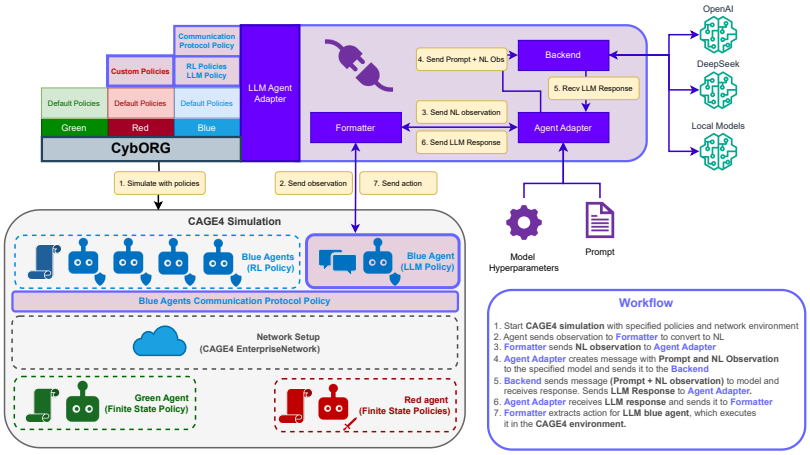

Fig. 2: Overview of our LLM Adapter Framework

blue agent is LLM-driven and the others are RL-driven. As shown, we add custom policies for both red and blue agents, we include an RL policy based on KEEP GNN for training and testing [16], we add a novel communication protocol for the blue agents, and propose a flexible LLM adapter to interact with diverse LLMs for action selection.

B. Observation and Response Formatting

The network state must be formatted in natural language to use an LLM for network defense. Since CybORG is designed for RL, it uses nonhuman-readable observation vectors. To overcome this situation, we built a custom wrapper that adapted CybORG's step observations by parsing them into natural language with the relevant information that each agent sees about their network.

Table II shows our observation format per step, which includes the name of the **Agent**, **Mission Phase**, the **Last Action** executed in previous steps and its **Status** result, the **Communication Vectors** broadcasted by blue agents, and a list of events under **Suspicious Activity Detected**.

Field	Value
Agent	`<agent_name>`
Mission Phase	`<value>`
Last Action	`<action><host/subnet>`
Last Action Status	`<TRUE/FALSE/UNKNOWN/IN_PROGRESS>`
Communication Vectors	`<list:binary_array>`
Suspicious Activity Detected	`<list:string>`

TABLE II: Formatted Observation for LLM agent

To extract the actions of the blue agent, we format the response of the LLM using a dictionary structure as seen in Fig. 3. If the LLM responds in an unexpected format that our action parser does not understand, we log an invalid action, and the agent will *Sleep* for that step. This helps us track when the model is not following the response structure we provided in the prompt.

C. Communication Protocol between Defender Agents

Since blue agents only have visibility in their assigned subnetwork (see Fig. 1), they need to exchange messages with each other to share threat information. CAGE 4 allows each agent to broadcast a 1-byte vector per step called *Communication Vector*, yet its format is undefined. We use this 8-bit protocol and propose a realistic multi-agent communication strategy.

Our idea is to summarize the current security level of a network based on each agent's observation and its current state (free or busy). This abstracts the network's general status without substantially increasing the observation dimensions. We propose that each agent should broadcast a message to the others notifying them of (1) suspicious activity detected from another agent's network, (2) the security level of its current network, and (3) its availability for running actions.

For remote activity detection (1), we assign each agent with one single bit since each agent can access only its own observation but can associate the origin of malicious activities from other networks. For the current security level (2), we provide more granularity by assigning 2 bits because the agent's observation contains more dimensions that describe the network security status in more detail. We assign only one bit for availability (3) since the agent can only have free and busy states.

Considering that defender agents are identified from 0 to 4, each blue agent i in the network creates their *Communication Vector* as follows:

- **Bits 0 to 4:** Set j bit to one if malicious action has been detected by i from an agent's j network. Otherwise, zero.
- **Bits 5 and 6:** Level of compromise that agent i identified in its subnetwork: `00` indicates no compromise, `01` netscan/remote exploit detected, `10` user-level compromise in a host, `11` admin-level compromise in a host.
- **Bit 7:** Set to one if agent i is waiting for an action to finish the execution. Otherwise 0.

This protocol can be extended to more detailed summaries depending on capacity and observation features. We believe

this approach provides sufficient information to agents to improve their action selection process, especially when agents notify a high-risk security level and other agents detect connections from their network.

D. Prompting

We aim to evaluate how LLMs act and reason in realistic incident response situations instead of maximizing the CAGE 4 reward function. Therefore, we design prompts that give the LLM a realistic view of a network and avoid mentioning that this as a game, a simulation, or having a reward function. Instead, we provide the agents the role of a "cybersecurity expert defending a network" with formatting rules and examples to guide the reasoning process. **The detailed prompt is in our repository.**

To evaluate the best prompting strategy that is realistic and minimizes penalties, we use a default CAGE 4 Finite State *red agent*, we set only one *blue agent* as an LLM-driven agent and the rest with a default CAGE 4 defender strategy. We start with an initial *Instructional Prompting* [20] where we only describe the task and the answer format. Then, we consider the *Few-Shot* strategy where, on top of the task description, we add examples of possible correct answers, which reduces invalid actions. Finally, we explore *Role Prompting* [21] by assigning a "cybersecurity expert" role to the agent.

Model	Role	Fewshot Instruct	Instruct
GPT-3.5 Turbo	-4307	-4349.5	-4620
4o-mini	-3810	-3219	-2888
o1-mini	-3022	-3243.5	-3890.5

TABLE III: Reward-based prompt tuning with OpenAI Models. Results are average rewards for blue agent.

As seen in Table III, on average, adding a role to the prompt with few-shot examples may increase the reward with different experiments. Therefore, we decide to use a prompt with a role and few-shot examples.

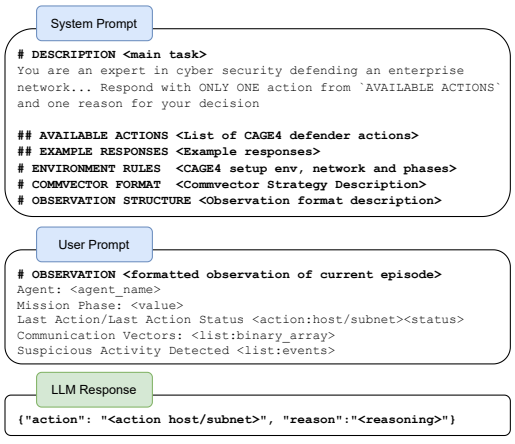

Fig. 3: Conversation Workflow for LLM Adapter

Figure 3 shows an overview of the interaction between the agent and the LLM. We follow the conventions of *system/developer* and *user* messaging for LLMs [19], [22], [23]. We set the *System Prompt* to describe the task, response validation, environment rules, and examples. The *User Prompt* contains the formatted observation per step. Note that the *System Prompt* remains consistent during all episodes, whereas the *User Prompt* might change due to the agent's observations.

We fine-tuned the prompt to include clear and concise definitions of each action, including response examples. This is necessary since action names can have multiple meanings, which can mislead the reasoning models. For instance, when we did not define the *Remove* action in our experiments, the LLM assumed it disconnected the host instead of erasing malicious processes.

Our extensible CybORG adapter for LLMs works with the latest online and offline models. For online versions, our current adapter supports OpenAI [19] and DeepSeek [22] through OpenRouter [24]. For offline models, it supports open-source models from HuggingFace [25], including Meta's LLaMA [23] family of models.

IV. Evaluation

In this section, we describe our experiments and their results. We first describe the methodology and the considerations for the environment setup. Then, we discuss in detail our experiment results and our analysis.

Environment and models: We use the CybORG version included in the CAGE 4 challenge repository [13] including our custom LLM adapter. We use the latest OpenAI models [19] *GPT-4o-mini, o1-mini* and *o3-mini*, and the the latest DeepSeek model: *DeepSeek-V3* [22]. When running our experiments, we focus on light models to reduce inference time and performance overhead. For all our models, we set the temperature to 1 to avoid limitations of context dependency from the prompt since it is intentionally not optimized for the CAGE 4 scenario. To keep track of the experiment variables, including reward, standard deviation, and action selection, we use the platform *Wandb (Weights and Biases)* [26].

LLM Messaging: Since CybORG runs hundreds of steps per episode, and each agent must select an action per step, we use a single *system/user* message to interact with the LLM to reduce token consumption. As seen in Fig. 3, we use one *system* prompt that contains the task to run. We then include the observation of the current step as a *user* message. The LLM-driven agents send a combination of these two messages per step and receive a response in JSON format containing an action and a reason justifying its decision.

Scenarios: For all our experiments, we run 2 episodes of the game with 500 steps each. Each agent (red, blue, and green) selects an action in each step, and CybORG simulates its execution. We use the default CybORG green agent policy *EnterpriseGreenAgent* while customizing red and blue. In our experiments, we consider four scenarios for blue agents:

- **No blue agents:** All blue agents *Sleep* every turn. Only red and green agents act. This serves as a baseline for our evaluation.
- **All blue agents as LLM:** All blue agents are LLM-driven. They all use the same LLM and the messaging

strategy with the same *system* prompt. Their *user* may vary for each agent's observation and include the agent's name to avoid confusing the LLM.

- **All blue agents as RL:** All blue agents are RL-based. We train them by using our own implementation of the communication vector and Cybermonic's KEEP GNN PPO model [16], which was among the best submissions for the CAGE 4 competition [13]. This submission did not implement a Communication Vector.
- **1 LLM and 4 RL blue agents:** To balance the RL and LLM distribution, we protect 3 subnets with 1 LLM agent (i.e., *blue_agent_4*) and 4 subnets with 4 RL (KEEP) agents.

Red agent variants: CybORG includes limited, finite-state red agents. To reduce the reality gap and diversify the adversary simulation, we implemented the red agent strategies for multi-agent reinforcement learning by Singh et al. [14]. In particular, we include the following:

- **AggressiveFSMAgent**: Employs an aggressive service discovery action, rapidly scanning the network to identify potential targets. While this approach can quickly map out network services, it increases the likelihood of detection by defensive measures.
- **StealthyFSMAgent**: Utilizes stealthy service discovery actions, aiming to remain undetected while gathering information about the network. This method is slower but reduces the risk of triggering security alerts.
- **ImpactFSMAgent**: Prioritizes action that impacts critical services, focusing on disrupting essential network functions. By targeting high-value assets, the ImpactFSMAgent aims to maximize operational disruption.
- **DegradeServiceFSMAgent**: Prioritizes action that degrades services used by the green agents. This can cause the *GreenAccessService* action to fail.

A. Experiments

Now we discuss our experiments and analyze our results in terms of the *Performance*, *Reward*, *Reasoning for Action Selection*. Then, we compare the reasoning between an RL and an LLM agent.

Performance: Regarding general LLM performance, our results show that *GPT-4o-mini* had the more balanced tradeoff between running time and reward. In Fig. 4, we show the results of our 1LLM+4RL experiment with multiple models. Among the tested models, *o3-mini* achieved the highest reward with the slowest execution time, followed by the bigger *o1-mini* model, which had a similar execution time. *Deepseek-V3* showed the lowest performance in terms of the CAGE 4 reward function but was faster than *o1* and *o3*. Finally, *GPT-4o-mini* had the quickest execution time and but low reward.

Regarding running time, we compared the experiments where all agents were either RL-driven or all were LLM-driven. We did not consider the training phase for the RL-driven agents nor the prompt engineering for task description and observation formatting for LLM-driven ones. On average, our experiments were **45.2 seconds** long when all blue agents were RL-driven and **4704.6** seconds when all of them were LLM-driven with the fastest model we tested (i.e., *GPT-4o-mini*). This means that, without considering the time-consuming training phase for RL agents, RL-driven agents were approximately **104.1 times** faster than the LLM-driven agents when performing action selection.

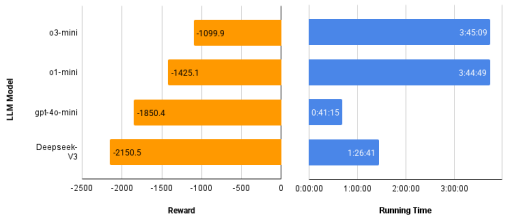

Fig. 4: Reward and running-time per model. 1 LLM-driven blue agent and others RL against default *FiniteState* red agent.

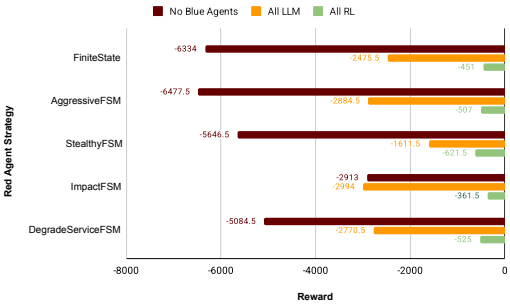

Fig. 5: Reward for each red agent strategy against LLM blue agents and RL blue agents.

Reward: The reward function for CAGE 4 is the cumulative joint reward for *blue agents*, based on penalties for service unavailability. From our experiments with 1LLM+4RL blue agents (Fig. 4), *o3-mini* received the least penalties. *GPT-4o-mini* had the lowest reward from the OpenAI models, but its execution time was the fastest of all the tested models. Since it is the most balanced, we compare *GPT-4o-mini* against different network agents and the KEEP RL approach.

For CAGE 4, the KEEP RL approach surpassed the all LLM-driven approach with *GPT-4o-mini*. Fig. 5 shows the reward value with different agents when there are no defenders, when all are LLM-driven (*GPT-4o-mini*), and when all are RL-driven (KEEP). Based on the reward *average* μ and *standard deviation* σ, an ALL RL team of defenders performs better against diverse red agents ($\mu = -493$, $\sigma = 95.9$) compared to an ALL LLM team ($\mu = -2547.2$, $\sigma = 498.8$). We see that the diversity of red agents did not substantially affect the reward for RL agents, contrary to the LLM-driven team of agents, which performed marginally better against one adversary (i.e., *StealthyFSM*).

The reward function definition is based on penalties on availability and assumes the attacker is always trying to deny the service to users. However, this is unrealistic since

attackers compromise systems without affecting availability [1]. For instance, with the *ImpactFSM* red agent, the *LLM Agent* performed worse compared to not having any defenders due to the attacker's goal (i.e., prioritize privilege escalation and impact a few compromised hosts). For this scenario, we believe the reward function did not correctly reflect the security level of the network since the adversary had high privileges on the network.

Reasoning for Action Selection: We also want to understand the reasoning behind the LLM agent's action selection. In particular, we compare the RL and LLM agents' action selection under similar scenarios. We first analyze the experiments involving an LLM with the best reward and performance (1 LLM + 4 RL agents, *o3-mini)*. We analyze one episode of 500 steps and remove the initialization step sample.

We generate clusters of similar reasons associated with their action to analyze them systematically. We use OpenAI's *text-embedding-3-large* model to convert textual data into numerical embeddings, followed by a PCA-enhanced K-Means [27] clustering strategy. Due to the high dimensionality of our data (499 samples with 3072 features each), DBSCAN [28] methods were not optimal for our analysis so we used the Elbow Method [29] and the Silhouette Score [30] and determined $K = 4$ as the optimal number of clusters. The application of PCA [31] helps us reduce the dimensionality to three components, facilitating its visualization and interpretability. Fig. 6 shows the generated clusters.

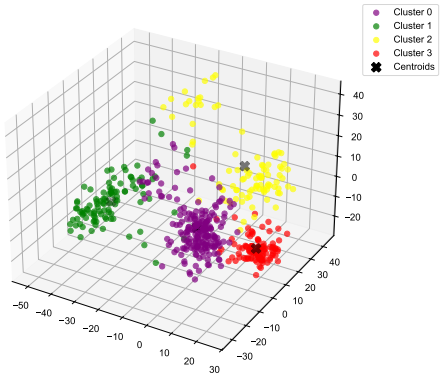

Fig. 6: Action Selection Clustering by Reason

The clusters and data points (DP) associated with the 499 samples are described in Table IV. We structured the action-reason pairs within each cluster into a dictionary format, then used GPT-4o to summarize each cluster and asking the model to "Summarize the main theme of the following security-related cluster in one sentence". We also manually analyzed each cluster to provide a custom description for each.

From a security standpoint, the reasoning for each cluster behind the action selection demonstrates a structured defensive strategy by the LLM-driven agent. Cluster 0 demonstrates a proactive defensive approach by executing analysis and deception actions when no suspicious activity has been detected, and the agent is available. Cluster 1 focuses on actions after receiving alerts and proactively attempts to remove malicious processes if another agent notifies of malicious activity. Cluster 2 shows how the agent tries to understand the failure caused by a previous action by analyzing it. Cluster 3 summarizes the agent's attempts to guarantee the correct deployment of techniques.

Cluster	GPT-4o Summary	Manual Summary
Cluster 0 (202 DP)	Proactively deploying decoys in various network zones is recommended as a precautionary measure to identify and lure potential red activity early, ensuring detection without impacting service availability amidst ongoing analysis and absence of current suspicious activity.	*DeployDecoy* as a preventative measure and run *Analyse* to detect malicious activity when no alerts are observed.
Cluster 1 (103 DP)	The main theme of the security-related text cluster is the need for further analysis of repeated INFO-level suspicious connections on various hosts and servers within a network to determine the presence and extent of potential 'red' activity or compromise.	*Analyse* and *DeployDecoy* when perceiving recurrent INFO-level connections; *Remove* as a preventative measure when other blue agents notify malicious activity.
Cluster 2 (93 DP)	The main theme of the text is that multiple failed decoy deployments in the office network indicate potential configuration issues or undetected threats, necessitating further analysis and reattempts to ensure robust detection and early warning of any red activity.	Attempts to run *DeployDecoy* or *Analyse* have failed. Run *Analyse* to detect configuration issues or malicious activity causing the failure.
Cluster 3 (101 DP)	The main theme of the text is understanding the importance of continuous monitoring and analysis of decoy deployments in progress to ensure their correct setup and effectiveness in detecting any early red agent activities while maintaining system security and integrity.	*DeployDecoy* action is in progress; then, run *Analyse* to validate the decoy is deployed correctly and reveal early red activity.

TABLE IV: Comparison of GPT-4o Summaries with Manual Summaries

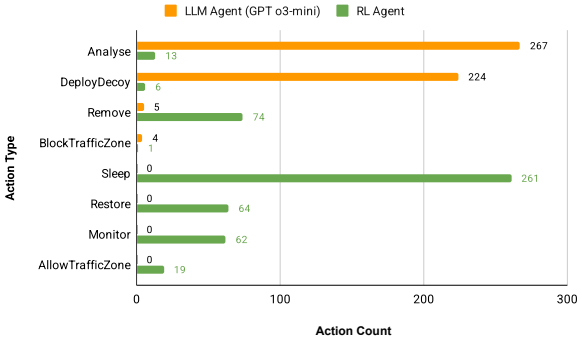

Fig. 7: Action Count Comparison for RL and LLM Agents

RL and LLM Action Selection Comparison: We compare the action selection process for an LLM-driven agent with the best reward (*o3-mini*) vs. a KEEP RL-driven agent against the default CAGE 4 attacker. Fig. 7 shows the summary of the action selection for one episode for each agent with a similar

scenario.

The KEEP RL agent follows a passive decision-making strategy, relying heavily on *Monitor*, *Sleep*, and *Remove* actions. It frequently transitions into *Sleep*, with a high probability of returning to this state from nearly every other action. When responding to potential threats, the agent prioritizes analyzing hosts for suspicious activity before committing to action. Actions like *Restore* and *BlockZoneTraffic* appear to be last-resort measures, likely due to their significant impact on availability. By avoiding aggressive interventions unless necessary, the agent ensures smooth network traffic flow and prevents disruptions.

On the other hand, the LLM agent follows a deception and analysis approach, prioritizing the deployment of decoys while avoiding *Restore* actions. After analyzing the reasoning log, we noticed that the LLM-driven agent decided to use *Remove* and *BlockTrafficZone* when other agents notified a possible compromise through their communication vector. In situations where a user-level compromise was detected elsewhere in the network, the agent prioritizes less disruptive actions like *DeployDecoy* or *Analyse* over *BlockTrafficZone*. The agent avoids the execution of the *Restore* action, possibly because it might cause availability issues for the green users. Our analysis showed that in some cases, the reason for action selection was prone to hallucinations; we saw multiple instances in which the agent misinterpreted the communication vector for an agent (i.e., assumed it was from agent 4 when it was from agent 3) or changed the definition of the action.

Here, we describe some of the hypotheses we have that justify the different action selection results between RL and LLM agents:

- According to the CAGE 4 documentation [13], the *Monitor* action is run automatically every turn, and calling it has no effect. Therefore, even though the RL agents selected it, we did not define it as an option in our prompt.
- To maintain the agent active, we did not define the *Sleep* action on the prompt. Including the *Sleep* definition and its use cases when the agent is busy could improve the performance of the LLM agents when actions are in progress.
- Our definition of *Restore* and *BlockTraffic* zone may have induced the LLM to avoid them to preserve availability. We described how these actions could disrupt user service. Therefore, adjusting the definitions of actions could diminish the availability of penalties.
- To evaluate the inherent LLM reasoning capabilities for ACD, our prompt does not include a strategy with explicit action decision rules nor situation-specific guidance. We could improve the performance of the LLM ACD agent with guidance based on the RL reasoning.

As we see, both strategies reflect security reasoning. Despite receiving more penalties than the RL agent, we see that the blue LLM-driven agent could effectively communicate with other RL agents, parse their observations, and reason with a security standpoint without any previous training aside from the provided context in its prompt.

V. Discussion

Now, we discuss some limitations of LLMs interacting in multi-agent environments with other ACDs and discuss possible venues to address them.

Environment Compatibility: The CAGE 4 environment is designed for RL training. In general, it is not fair to ask a general purpose LLM to select an action without the context an RL-trained approach will have. We are synthesizing the high-dimensional observations of an RL defender for an LLM to reason about it, so an LLM will not observe and interpret the impact of its decisions as well as an RL-driven agent. Moreover, we are conditioning the success of the LLM agent to the reward function designed for the gym.

Hallucinations: Despite describing the simulation rules in our prompt, the LLM agents were prone to hallucinations, including misreading the communication vector observations, compromise levels, and security events. Increasing inference time and fine-tuning models could address this issue with an efficiency drop for action selection. LLM reasoning can improve fast with context-dependent prompting strategies such as Analogical Reasoning [32]. Still, it can be prone to hallucinations when the model has not been trained with contextually relevant exemplars.

Prompt Definition: Since our goal is to provide the agent with a realistic context, we design the prompt without specifying the reward policy or providing reward values to the LLM. Adjusting the prompt by describing the reward function used by the simulation and adding context to the simulation constraints (e.g., action execution steps, reward per step/episode, red agents strategies) will increase the expected reward. Still, it will impact the realism of the agent's reasoning for real security scenarios. The prompt can be improved to describe each action in more detail to avoid confusion, as the ones identified with the reason clusters 2 and 3. However, augmenting the length of the prompt will also increase the token consumption.

VI. Conclusions

We integrate a novel approach to integrate LLM reasoning for defender agents for CybORG's CAGE 4, a realistic ACD multi-agent environment based on RL. We evaluated them against diverse red agent strategies and compared the LLM defending reasoning against pre-trained RL in different scenarios. Our experiments show that LLM ACD agents can communicate effectively when they share a communication protocol. Together, they can protect a network with a security reasoning similar to a team of security operators.

Despite the discussed LLM limitations, we believe ACD gyms would benefit from LLM reasoning for defenders to bridge their reality gap for transferability and explainability. LLMs can also facilitate changing defensive strategies without retraining by prompt tuning. Defenders can benefit from LLM integrations to RL ACD gyms to train realistic ACD agents' security reasoning without the risks of deploying them on real networks. Our LLM implementation for CybORG facilitates the transferability of optimal policies to other ACD and real

environments. This could be done through prompt tuning to reproduce optimal pre-trained RL policies.

ACKNOWLEDGMENTS

This material is based upon work supported in part by the Air Force Office of Scientific Research under award number FA9550-24-1-0015, and by the National Center for Transportation Cybersecurity and Resiliency (TraCR) USDOT Grant #69A3552344812. For this research, we also collaborated with OpenAI through the Cybergrant program, using their API credits and funding to advance our research in AI-driven cybersecurity solutions.

REFERENCES

[1] CheckPoint, "A Closer Look at Q3 2024: 75% Surge in Cyber Attacks Worldwide," 2024. [Online]. Available: https://blog.checkpoint.com/research/a-closer-look-at-q3-2024-75-surge-in-cyber-attacks-worldwide/

[2] S. R. Castro and A. A. Cárdenas, "Ghost in the SAM: Stealthy, robust, and privileged persistence through invisible accounts," in *Proceedings of the 2024 Workshop on Research on Offensive and Defensive Techniques in the Context of Man At The End (MATE) Attacks*, ser. CheckMATE '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 59–72. [Online]. Available: https://doi.org/10.1145/3689934.3690839

[3] M. Wolk, A. Applebaum, C. Dennler, P. Dwyer, M. Moskowitz, H. Nguyen, N. Nichols, N. Park, P. Rachwalski, F. Rau, and A. Webster, "Beyond cage: Investigating generalization of learned autonomous network defense policies," 2022. [Online]. Available: https://arxiv.org/abs/2211.15557

[4] M. Standen, M. Lucas, D. Bowman, T. J. Richer, J. Kim, and D. Marriott, "CybORG: A Gym for the Development of Autonomous Cyber Agents," Aug. 2021.

[5] A. Kott, "Autonomous intelligent cyber-defense agent: Introduction and overview," 2023. [Online]. Available: https://arxiv.org/abs/2304.12408

[6] P. Theron, "Alternative architectural approaches," in *Autonomous Intelligent Cyber Defense Agent (AICA): A Comprehensive Guide*, ser. Advances in Information Security, A. Kott, Ed. Springer International Publishing, 2023, vol. 87, pp. 17–46.

[7] J. Loevenich, E. Adler, T. Huerten, and R. Rigolin Ferreira Lopes, "Design and evaluation of an autonomous cyber defence agent using drl and an augmented llm," 2024, available at SSRN: https://ssrn.com/abstract=5076836 or http://dx.doi.org/10.2139/ssrn.5076836. [Online]. Available: https://ssrn.com/abstract=5076836

[8] G. Palmer, C. Parry, D. J. B. Harrold, and C. Willis, "Deep reinforcement learning for autonomous cyber defence: A survey," 2024. [Online]. Available: https://arxiv.org/abs/2310.07745

[9] A. Lohn, A. Knack, A. Burke, and K. Jackson, "Autonomous cyber defence: a roadmap from lab to ops," *Center for Emerging Technology and Security*, 2023.

[10] C. Seifert, M. Betser, W. Blum, J. Bono, K. Farris, E. Goren, J. Grana, K. Holsheimer, B. Marken, J. Neil, N. Nichols, J. Parikh, and H. Wei, "Cyberbattlesim," https://github.com/microsoft/cyberbattlesim, 2021.

[11] J. Janisch, T. Pevný, and V. Lisý, "Nasimemu: Network attack simulator & emulator for training agents generalizing to novel scenarios," in *Computer Security. ESORICS 2023 International Workshops: CPS4CIP, ADIoT, SecAssure, WASP, TAURIN, PriST-AI, and SECAI, The Hague, The Netherlands, September 25–29, 2023, Revised Selected Papers, Part II*. Berlin, Heidelberg: Springer-Verlag, 2024, p. 589–608. [Online]. Available: https://doi.org/10.1007/978-3-031-54129-2_35

[12] A. Molina-Markham, C. Miniter, B. Powell, and A. Ridley, "Network environment design for autonomous cyberdefense," https://arxiv.org/abs/2103.07583, 2021.

[13] TTCP Working Group, "Ttcp cage challenge 4," https://github.com/cage-challenge/cage-challenge-4, 2023.

[14] A. V. Singh, E. Rathbun, E. Graham, L. Oakley, S. Boboila, A. Oprea, and P. Chin, "Hierarchical Multi-agent Reinforcement Learning for Cyber Network Defense," https://arxiv.org/abs/2410.17351, Oct. 2024.

[15] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347

[16] Cybermonic, "KEEP: A GNN-based PPO Model for MARL," 2025, available: `https://github.com/cybermonic/cage-4-submission`. [Online]. Available: https://github.com/cybermonic/cage-4-submission

[17] M. Rigaki, O. Lukáš, C. Catania, and S. Garcia, "Out of the cage: How stochastic parrots win in cyber security environments," in *Proceedings of the 16th International Conference on Agents and Artificial Intelligence*. SCITEPRESS - Science and Technology Publications, 2024, p. 774–781. [Online]. Available: http://dx.doi.org/10.5220/0012391800003636

[18] Y. Yan, Y. Zhang, and K. Huang, "Depending on yourself when you should: Mentoring llm with rl agents to become the master in cybersecurity games," 2024. [Online]. Available: https://arxiv.org/abs/2403.17674

[19] OpenAI, "OpenAI Models and Research," 2024, available: `https://openai.com`. [Online]. Available: https://openai.com

[20] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language Models are Few-Shot Learners," in *Advances in Neural Information Processing Systems*, vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901.

[21] A. Kong, S. Zhao, H. Chen, Q. Li, Y. Qin, R. Sun, X. Zhou, E. Wang, and X. Dong, "Better Zero-Shot Reasoning with Role-Play Prompting," in *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*. Mexico City, Mexico: Association for Computational Linguistics, 2024, pp. 4099–4113.

[22] DeepSeek, "DeepSeek: Advanced AI Language Models and Enterprise Solutions," 2025, available: `https://deepseek.ai`. [Online]. Available: https://deepseek.ai

[23] Meta AI, "LLaMA: Open and Efficient Foundation Language Models," 2023, available: `https://ai.meta.com/llama`. [Online]. Available: https://ai.meta.com/llama

[24] OpenRouter, Inc., "OpenRouter: A Unified Interface for Large Language Models," 2025, available: `https://openrouter.ai`. [Online]. Available: https://openrouter.ai

[25] Hugging Face, "Hugging Face: Democratizing Machine Learning," 2024, available: `https://huggingface.co`. [Online]. Available: https://huggingface.co

[26] Weights & Biases, "Weights & Biases: Machine Learning Experiment Tracking," 2025, available: `https://wandb.ai/site/`. [Online]. Available: https://wandb.ai/site/

[27] P. Berkhin, "K-means clustering," in *Encyclopedia of Machine Learning*, C. Sammut and G. I. Webb, Eds. Springer, 2011, pp. 563–564. [Online]. Available: https://link.springer.com/referenceworkentry/10.1007/978-0-387-30164-8_425

[28] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, ser. KDD'96. AAAI Press, 1996, p. 226–231.

[29] H. Humaira and R. Rasyidah, "Determining the appropiate cluster number using elbow method for k-means algorithm," in *Proceedings of the 2nd Workshop on Multidisciplinary and Applications (WMA)*, 2020, pp. 1–8.

[30] P. J. Rousseeuw, "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis," *Journal of Computational and Applied Mathematics*, vol. 20, pp. 53–65, 1987. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0377042787901257

[31] W. J. Weber and J. C. Morris, "Kinetic models of sorption processes," *Journal of the Sanitary Engineering Division*, vol. 89, no. 2, pp. 31–60, 1963. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/009830049390090R

[32] M. Yasunaga, X. Chen, Y. Li, P. Pasupat, J. Leskovec, P. Liang, E. H. Chi, and D. Zhou, "Large Language Models as Analogical Reasoners," https://arxiv.org/abs/2310.01714, Mar. 2024.