Out of the Cage: How Stochastic Parrots Win in Cyber Security Environments

Maria Rigaki[1][a], Ondřej Lukáš[1][b], Carlos Catania[2][c] and Sebastian Garcia[1][d]

[1]*Faculty of Electrical Engineering, Czech Technical University in Prague, Czech Republic*

[2]*National Scientific and Technical Research Council (CONICET), Argentina*

Keywords: Reinforcement Learning, Security Games, Large Language Models.

Abstract: Large Language Models (LLMs) have gained widespread popularity across diverse domains involving text generation, summarization, and various natural language processing tasks. Despite their inherent limitations, LLM-based designs have shown promising capabilities in planning and navigating open-world scenarios. This paper introduces a novel application of pre-trained LLMs as agents within cybersecurity network environments, focusing on their utility for sequential decision-making processes. We present an approach wherein pre-trained LLMs are leveraged as attacking agents in two reinforcement learning environments. Our proposed agents demonstrate similar or better performance against state-of-the-art agents trained for thousands of episodes in most scenarios and configurations. In addition, the best LLM agents perform similarly to human testers of the environment without any additional training process. This design highlights the potential of LLMs to address complex decision-making tasks within cybersecurity efficiently. Furthermore, we introduce a new network security environment named NetSecGame. The environment is designed to support complex multi-agent scenarios within the network security domain eventually. The proposed environment mimics real network attacks and is designed to be highly modular and adaptable for various scenarios.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

1 INTRODUCTION

From text generation to summarization, LLMs have exhibited an exceptional capacity to replicate human-like linguistic capabilities. However, their potential extends beyond these conventional applications. Recently, LLMs have demonstrated planning and open-world exploration abilities, hinting at their potential to extend their original boundaries (Park et al., 2023).

One such domain where these emerging capabilities hold significant promise is cybersecurity. Automation of network security testing (penetration testing) has been part of the research agenda in the past, mainly centered around reinforcement learning (RL) agents and environments. Fusing LLMs with sequential decision-making processes introduces an interesting new exploration avenue.

This paper delves into the intersection of LLMs, cybersecurity, and sequential decision-making. We present a novel approach that uses pre-trained LLMs as agents within cybersecurity environments. By introducing LLM agents, we seek to explore whether these models can not only match but potentially outperform conventional RL agents in network security scenarios. To evaluate the effectiveness of our proposed approach, we tested it in two different security environments: Microsoft's CyberBattleSim (Microsoft, 2021) and our new network security environment named NetSecGame. In addition to the comparison with other RL-based agents, we performed experiments to select the best agent design and the best-performing pre-trained LLM.

Experiments showed that pre-trained LLM agents can succeed in different scenarios with win rates of 100% when there is no defender present and 50% when a defender is present in the most challenging scenario (80% win rate in the easier scenario). When comparing pre-trained LLMs, we found that GPT-4 (OpenAI, 2023) outperforms GPT-3.5-turbo significantly. The main contributions of the paper are:

- The use of pre-trained LLM agents designed for network cybersecurity scenarios. The agent's performance is comparable to or better than rein-

[a] https://orcid.org/0000-0002-0688-7752
[b] https://orcid.org/0000-0002-7922-8301
[c] https://orcid.org/0000-0002-1749-310X
[d] https://orcid.org/0000-0001-6238-9910

Rigaki, M., Lukáš, O., Catania, C. and Garcia, S.
Out of the Cage: How Stochastic Parrots Win in Cyber Security Environments.
DOI: 10.5220/0012391800003636
Paper published under CC license (CC BY-NC-ND 4.0)
In *Proceedings of the 16th International Conference on Agents and Artificial Intelligence (ICAART 2024) - Volume 3*, pages 774-781
ISBN: 978-989-758-680-4; ISSN: 2184-433X
Proceedings Copyright © 2024 by SCITEPRESS – Science and Technology Publications, Lda.

forcement learning agents that require thousands of training episodes.

- A new network security RL modular environment, called NetSecGame, that implements realistic conditions, including a defender.

2 RELATED WORK

2.1 LLMs for Planning and Reinforcement Learning

Pre-trained LLMs face challenges in long-term planning, occasionally leading to irrelevant or unhelpful actions. However, frameworks such as ReAct (Yao et al., 2023), Reflexion (Shinn et al., 2023), and DEPS (Wang et al., 2023b) demonstrated that LLM agents can become better planners through reasoning and self-reflection. ReAct (Yao et al., 2023) combines reasoning with action and excels in question-answering tasks that demand multiple logical steps. Reflexion (Shinn et al., 2023) introduces a sequential decision-making framework that incorporates self-reflection and evaluation components to assess the quality of actions the agent takes and short-term and long-term memory. In this work, we used the ReAct agent architecture in the NetSecGame environment (4.2) which provided good results while retaining speed and simplicity compared to other frameworks such as Reflexion and DEPS. In non-security environments, LLM agents have showcased exploratory capabilities (Wang et al., 2023a; Du et al., 2023; Wu et al., 2023) in gaming environments such as Minecraft and Crafter. While these environments require a series of actions to reach a goal, they are not adversarial, at least in the current studies.

2.2 Cybersecurity Reinforcement Learning Environments

Currently, there exist several environments for training and testing agents in network-based cybersecurity scenarios using RL principles (Elderman et al., 2017; Hammar and Stadler, 2020; Microsoft, 2021; Standen et al., 2021; Andrew et al., 2022; Janisch et al., 2023). One of the main issues with prior work is that the authors of each environment make different decisions about network behavior and goals, the presence of a defender or not, and how rewards are counted. Even though these decisions are essential to determine if an agent can be used in a real network, most environments do not discuss or justify them in detail. Regarding scalability, most environments support the OpenAI Gym (Brockman et al., 2016) API, enabling off-the-shelf RL libraries and algorithms to train the agents. However, these environments mostly rely on naive vectorization of the state space using adjacency matrices plus additional feature vectors to hold information about services, versions, and possible exploits for each service. Given the amount of hosts, the services that are running within an entreprise environment, their respective versions, and the applicable exploits, can lead to a combinatorial explosion of the state vector's dimensionality. We believe that the state and action representation of cyber security environments in a large scale is not a currently solved problem.

3 NetSecGame

NetSecGame, our innovative simulated network security environment for training and testing attack and defense strategy agents, is accessible through a public repository [1]. Distinguished from previous work, it aligns more closely with actual attacks by offering modularity for easy topology extension, restricting agent information to what an actual attacker would receive, employing a realistic goal of exfiltrating data to the Internet, introducing a defender, and utilizing generic, non-engineered rewards. Following a reinforcement learning model, agents interact with NetSecGame through a Python API, engaging in actions and receiving new states, rewards, and end-of-game signals. The environment's configurability spans diverse network topologies, encompassing hosts, routers, services, and data. NetSecGame encapsulates six main components: (i) configuration, (ii) action space, (iii) state space, (iv) reward, (v) goal, and (vi) defensive agent, aiming to provide a realistic yet high-level depiction of network security attacks.

3.1 Configuration of NetSecGame

NetSecGame uses two configuration files—one for defining the network topology and another for defining the behavior of the environment.

The network topology configuration uses a configuration file from the CYST simulation environment (Drašar et al., 2020). CYST was used since it is a flexible simulation engine based on network events. Different configuration files for the topology define different 'scenarios' as described in Subsection 3.1. The network topology configuration file defines Clients, Servers, Services, and Data.

[1] https://github.com/stratosphereips/NetSecGame/

The second configuration file determines the initial placement of the agent, as well as if a defender is present or not, the specific *scenario* used, the maximum amount of actions allowed (steps), and for each action, the probability of success and the probability of detection (if there is a defender present).

NetSecGame includes the option to have a defender in the environment that represents the concept of a security operations team that has visibility of the whole network. The agent is called *StochasticDefenderWithThreshold* and detects repeated actions using a probabilistic approach combined with various thresholds per action type and time interval.

Network Scenarios. NetSecGame offers three predefined network scenarios of increasing complexity, each differing in the number of clients, servers, services, and data. However, its extensibility allows easy customization. The attacker's goal in each scenario is defined as a specific state, with victory achieved upon reaching that state undetected. This goal-setting enables users to define diverse objectives; for instance, discovering a particular service becomes a winning state when linked to a host. NetSecGame's flexibility allows randomizing network elements such as IP addresses and data positions, and, crucially, randomizing the goal per episode. This feature is essential for both human players and agents, preventing the establishment of repetitive patterns in human gameplay and testing the adaptability of agents to diverse scenarios. In simulated environments, where real attackers may attack only once on the same network, randomization ensures a fair and dynamic gaming experience.

State Representation. NetSecGame represents states as a collection of assets known to the attacker: *known networks*, *known hosts*, *controlled hosts*, *known services*, and *known data*. Note that the agent can compute all this data, and the environment only facilitates it. There is no extra help in understanding the environment. After each action, the agent receives a new state of the environment. This design is based on the fact that the attackers often have limited knowledge about the network and gradually discover it throughout interactions.

Action Representation. Currently, NetSecGame only supports attacker agents and actions (the defender is not an agent). Actions define the transition between states. There are five types of actions available, each with parameters: *ScanNetwork*, *FindServices*, *ExploitService*, *FindData*, and *ExfiltrateData*. The list of valid actions is never sent. The agents determine which valid actions based on the current state.

Reward Function. The reward function in NetSecGame consists of three non-exclusive parts. First, there is a reward of -1 for taking any step in the environment. Second, the reward for reaching the goal, which results in the termination of the episode, is 100. Last, when the defender detects the agent, which also terminates the episode, it is awarded with -50. No rewards are given for intermediate states.

Differences with Existing Security Environments. The main differences between NetSecGame and other environments are the design based on real attacks and to run the agents in real networks in the future. In particular:

- The network topologies represent small to medium organizations. Clients and servers are in separate networks, with one connection to the Internet.
- The parameters for each action are not sent to the agent. The basic actions are known, but the action space is **not** sent to the agents. NetSecGame is then incompatible with the Gym environment, but it is more representative of what an attacker knows. Other environments send to the agent the complete set of valid actions.
- The goal of NetSecGame is to exfiltrate data as an APT attack. Other environments' goals are 'controlling more than half the network', which is not representative of real attackers.
- NetSecGame has an internal defender that detects, blocks, and terminates the game.

4 LLM AGENTS FOR NetSecGame

LLM agents are similar to other RL agents in their interaction with the environment. At time t, the agent receives the state s_t and the reward r_t, processes the state and proposes a new action a_{t+1}. The main differences between LLM and traditional agents are that they use a textual representation of the state and that they do not learn a policy. They select actions based on the knowledge accumulated during training and by using prompt techniques such as "one-shot" learning.

4.1 Single-Prompt Agents

These agents have a single prompt and a simple memory. The prompt had multiple elements, such as system instructions and rules, a list of previous actions (memory), a text representation of the state s_t, a "one-shot" example of each action, and the text *query* ask-

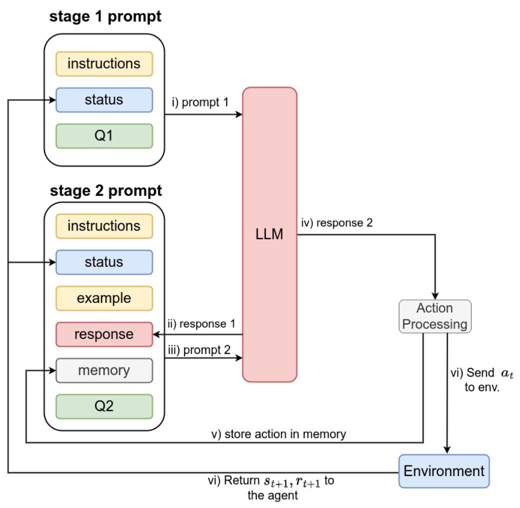


Figure 1: The ReAct agent prompt structure and workflow.

ing for the next action. Detailed prompts can be found in the public repository.

4.1.1 *Temperature* Variant

The *temperature* variant of the single-prompt agent uses three memory strategies to prevent repetitions. These include maintaining a list of the last k non-repeated actions (*memory-a*), a list of repeated actions (*memory-b*) with counts, and presenting the previous action separately in the prompt (*memory-c*). The temperature variant prompt includes initial system instructions, rules, the last k non-repeated actions, repeated actions, the current state s_t, an example of each action, the last action taken (*memory-c*), and a query to select the best action. For certain pre-trained LLMs, the memory strategy alone may not prevent action repetition. To address this, the agent adjusts the LLM's temperature parameter based on the number of repeated actions in the last k actions. This pushes the LLM to generate more diverse outputs.

4.2 ReAct Agent

The ReAct design is used for NetSecGame in two stages. First, the agent asks the LLM to reason about the environment's state; second, the LLM is asked to select the best action. Figure 1 shows the prompts' structure and workflow. The first stage has:

1. **Instructions** and rules about the environment.
2. A textual representation of the **state** s_t.
3. A query to evaluate the status and the possible actions (**Q1 prompt**):

```
List the objects in the current status and
the actions they can be used. Be specific.
```

The second stage has:

1. **Instructions** and rules about the environment.
2. A textual representation of the **state** s_t.
3. One **example** for each possible action.
4. The **response** to the first stage prompt.
5. A list of the last k actions (**memory**)
6. A query to select the best possible action (**Q2 prompt**):

```
Provide the best action and its parameters
in the correct JSON format. Action:
```

If the processed response in the proposed action a_t is valid, it is sent to the environment. The action and its validity are stored in the memory, along with an indication of usefulness. Since the environment offers sparse rewards, an action is labeled useful if the environment's new state s_{t+1} differs from the previous one. This is a form of agent's *intrinsic* reward. The new state s_{t+1} is used to construct the prompts for the next step. The process continues until either the goal is reached, the maximum steps are reached, or the agent is detected. Examples of the full prompts are presented in the public repository.

SCIENCE AND TECHNOLOGY PUBLICATIONS

5 LLM AGENT FOR CyberBattleSim

The LLM agent designed for the CyberBattleSim environment follows a single-prompt approach, as detailed in Section 4.1, with modifications to the prompt component order. Due to a distinct goal and action space in the chain scenario, the agent necessitates different instructions and rules. In initial experiments, an issue arose where the agent would stop searching after discovering all nodes except the last one. To address this, we introduced a prompt that tricks the LLM into perceiving more than 20 hosts in the network.

```
You are a competent pentester. There are
more than 20 computers in the network.
Your goal is to discover and own
all of the computers.
```

The final prompt consists of the following five components: the initial system instructions, a "one-shot" example of each action, a list of the last k actions (memory), a textual representation of the state

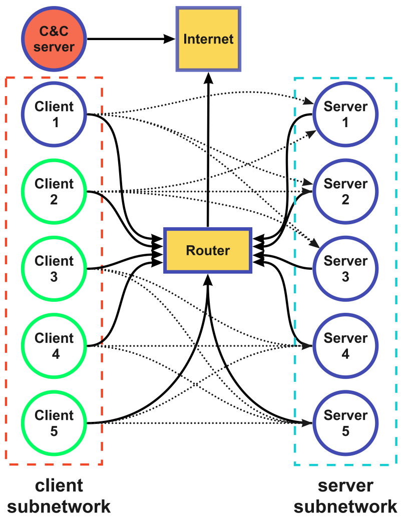


Figure 2: Setup of both NetSecGame topology versions: small scenario with the blue parts and full scenario in green.

s_t, the rules of the game, and the query for selecting the next action. Since the single-prompt approach performed well in our experiments (Section 7.4), we decided not to design and test a ReAct LLM agent.

6 EXPERIMENTAL SETUP

6.1 NetSecGame Configuration

Figure 2 shows the two scenarios we used in NetSecGame ("small" and "full"). The "small" scenario has five servers, one client in a separate network, a main router connecting both networks, and an Internet router providing access to an external C&C host for data exfiltration. The servers have one or two services each, while the clients have one. Data quantities on servers vary from three to zero. The "full" scenario mirrors the small one but includes five clients.

In all experiments, the goal was to exfiltrate specific data to the C&C server. Success requires the attacker to discover hosts and services, exploit services, locate data, and transmit them to the correct server. The LLM agents operated in dynamically configured environments with randomized networks, IP addresses, and data locations. The smaller scenario was designed for testing strategies, while both were used to compare the best LLM agent against baselines, with and without a defender. All LLM agents experiments were repeated 30 times with $max\_steps = 30$, then 60, and 100. Each episode is independent.

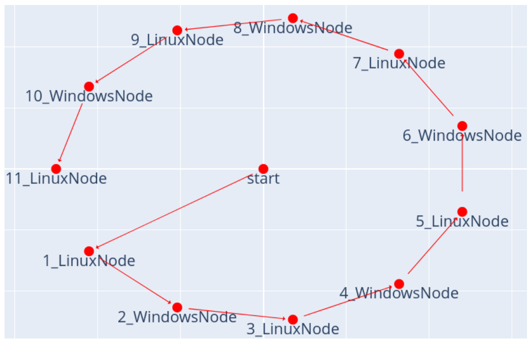


Figure 3: Network topology of the chain scenario in CyberBattleSim solved with the minimum amount of actions.

6.1.1 Baselines

For the baseline comparisons, we selected a random agent, a random agent with a no-repeat heuristic, and a tabular Q-learning agent (Watkins and Dayan, 1992). We ran five trials for each baseline, averaging results. The Q-learning agent was trained for 50,000 episodes in all scenarios while the random agent experiments were run for 2,000 episodes.

6.2 CyberBattleSim Environment

CyberbattleSim provides three scenarios, and we chose the "chain" scenario (Figure 3) with ten nodes due to its complexity and distinct goals compared to NetSecGame. Agents must traverse ten nodes to reach the final host, using local or remote attacks, including a "connect and infect" action. Positive rewards are given for owning a new host, discovering credentials, and reaching the final host. Negative rewards penalize repeated attacks, failed exploits, and invalid actions. CyberBattleSim features an "interactive mode" for human or Python program interaction, used by the LLM agent. Our tests found discrepancies between the interactive mode and the Gym implementation, where the negative rewards were removed from the Gym environment. This allows Gym agents to perform actions without costs. We decided to keep the negative rewards in all environments.

6.2.1 Baselines

The baseline agents for the CyberbattleSim tests were a random agent, a random agent with a heuristic that greedily exploits any credentials found, and a Deep Q-learning Network (DQN) agent (Mnih et al., 2013). All agents used 100 max iterations per training episode. The DQN agent was trained for 50 training episodes. All agents were evaluated in 10 episodes.

Table 1: Average win rates and returns of all LLM agents in the small scenario (randomized target) with 60 max_steps and 30 episodes. The asterisk indicates that some episodes were not computed due to issues with the OpenAI API.

	GPT-3.5 turbo		GPT-4	
Agent	Win Rate	Return	Win Rate	Return
S-Prompt	0.0%	-100.0	100.0%*	78.4
S-Prompt(T)	26.67%	-24.8	43.33%	3.0
ReAct	33.33%	-13.3	100.0%	83.1

7 RESULTS

7.1 LLM Agents Comparison

Table 1 shows the win rates ($\frac{wonepisodes}{\#episodes}$) and returns for the single-prompt and ReAct agents using both GPT-3.5-turbo and GPT-4 in the small NetSecGame scenario without a defender. The single-prompt GPT-4 agent was stopped in two of the 30 runs due to the OpenAI API's rate limitations. This happened before the expiration of the 60 max_steps, which means that the agent may have had a slightly lower win rate.

There is a large difference between GPT-4 and GPT-3.5-turbo since the latter repeats actions. An agent with variable temperature was created to solve this problem, improving from 0 to 26% win rate. However, the design did not work well with GPT-4. The ReAct architecture works well with GPT-4, and it improves the GPT-3.5-turbo win rate from 0 to 33%. The ReAct agent is more stable than the Single-Prompt agent, requiring fewer steps on average in an episode. Therefore, it was used for the subsequent experiments.

7.2 NetSecGame Small Scenario

The win rates of the baselines and the ReAct agent in the small scenario with and without a defender are presented in Figure 4. The figures show the results in different max_steps settings. Without a defender, the ReAct agent wins 100% of the time in the 60 and 100 max_steps setting and outperforms the baselines. When the max steps are limited to 30, it wins 80% of the time, which is still the best performance. The random agent with the no-repeat heuristic shows that, given enough steps, it eventually wins.

Table 2 shows the average returns and detection rates on the small scenario with 60 max_steps. The average returns show a similar view as the win rates. The lowest detection rate is reported by the random no-repeat agent (15.81%), with the ReAct agent closely following at 16.67%

7.2.1 Human Performance

We also conducted tests with eight human experts playing the game in interactive mode, resulting in 22 sessions. Despite the informal evaluation, it gave insights into the performance of agents. Without a defender, humans solved the small environment in an average of 17.68 moves and an average return of 82.32, comparable to the ReAct agent's performance. Humans found patterns in the environment, such as a relevant subnet, leading to more efficient solutions.

7.3 NetSecGame Full Scenario

Figure 4 shows win rates in the full scenario with and without a defender for different max_steps. Without a defender, the ReAct agent wins 100% of the time using 60 and 100 max_steps. With a defender, the Q-learning agent has the best performance, and it seems that the detections helped the agent learn a good policy. This highlights that learning from the past can prevent "bad" behaviors.

The ReAct agent has a winning rate of 50% for $max\_steps >= 60$ and positive returns. (Table 3). None of the prompts has instructions to avoid the defender. The ReAct agent sometimes follows a breadth-first approach, scanning hosts for services, which can trigger the defender.

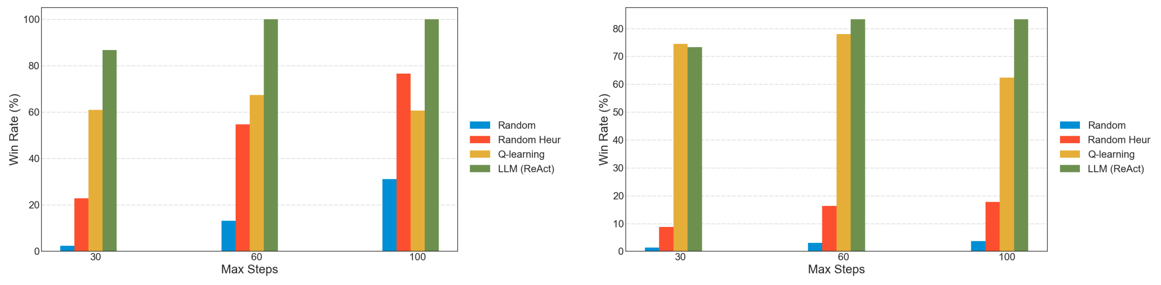
7.4 CyberBattleSim Chain Scenario

Table 4 presents results for win rate, return, and episode steps for agents in the "chain" scenario (averages over ten runs). The LLM agent with GPT-4 and a simple "one-shot" prompt won all runs with few steps. The DQN baseline also won all trials. The random agents won only if the number of maximum steps was higher than 1,000, while the LLM and DQN agents performed well with 100 steps.

A "quirk" of the "chain" scenario is that the minimum number of steps to solve the game is 22 (return of 6,154). However, agents can score higher by performing 'unnecessary' actions with a positive reward.

8 LIMITATIONS AND FUTURE WORK

We discovered several limitations during the design and experimentation of LLMs as agents. GPT-3.5 hallucinated, proposing actions with unknown objects. Additionally, it repeated invalid actions in a verbose manner and deviated from the format. The cost of using the GPT-4 API was substantially higher, being 30

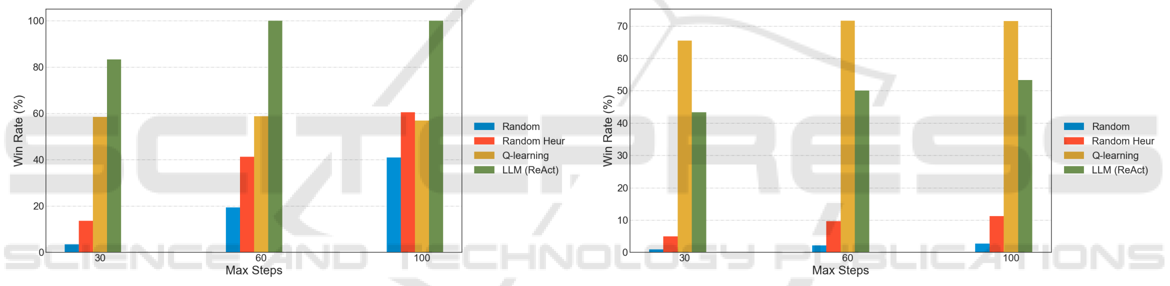


(a) No defender. (b) With stochastic-threshold defender.

Figure 4: For the NetSecGame small scenario, win rates for different numbers of max_steps.

Table 2: NetSecGame small scenario: average win rates, returns, and detection rates of all agents with random target per episode. With a maximum of 60 max_steps per episode and 30 episodes of repetition in LLM-based agents.

	No Defender		Defender		
Agent	Win Rate	Return	Win Rate	Return	Detection Rate
Random	13.21%	-37.18	2.99%	-64.30	18.68%
Random (no-repeat)	54.76%	8.47	16.28%	-43.49	**15.81%**
Q-learning	67.41%	47.55	77.96%	54.91	16.28%
ReAct	**100.0%**	**83.10**	**83.33%**	**58.83**	16.67%



SCITEPRESS
SCIENCE AND TECHNOLOGY PUBLICATIONS

(a) No defender. (b) With stochastic-threshold defender.

Figure 5: For the NetSecGame full scenario, win rates in different numbers of max_steps.

Table 3: Avg returns and detection rates of agents in the full scenario with random target per episode. With 60 max_steps per episode and 30 episodes of repetition in LLM-based agents.

	No Defender		Defender		
Agent	Win Rate	Return	Win Rate	Return	Detection Rate
Random	19.43%	-44.46	2.18%	65.11	93.95%
Random (no-repeat)	41.32%	-9.19	9.63%	-52.96	83.63%
Q-learning	58.74%	48.0	**71.0%**	**45.38**	**24.58%**
ReAct	**100.0%**	**77.13**	50.0%	8.20	43.33%

Table 4: Average win rate, return, and episode steps of all agents in the chain scenario of CyberBattleSim.

Agent	Win Rate	Return	Episode steps
Random	0.0%	-726.98	100.0
Random (cred.)	0.0%	-998.25	100.0
DQN	100.0%	6154.2	22.3
LLM	100.0%	6160.7	31.0

times more expensive than GPT-3.5. GPT-4 is currently the only model capable of handling multiple scenarios without fine-tuning. Open-source models will be fine-tuned next.

The instability and evolution of commercial models made it hard to reproduce results. The art-like nature of prompt creation is challenging, as small changes impact model behavior, making evaluation complex. Current agents do not learn from past episodes, a feature to incorporate later. For NetSecGame, future work includes adding a trainable defender and adding multi-agent capabilities.

9 CONCLUSIONS

This work designed agents with pre-trained LLMs to solve cybersecurity environments. The LLM agents solved two security environments without additional training steps and without learning between episodes, differing from traditional RL agents that require tens of thousands of training episodes.

Pre-trained LLMs have limitations and costs, including shortcomings in reproducing the results of black-box commercial models. However, there is potential in using LLMs for high-level planning of autonomous cybersecurity agents. Future work will focus on more complex scenarios and environments.

NetSecGame is designed to be realistic while providing a high-level interaction API for agents. It implements a modular configuration for topologies, a goal definition, and a reward system without leaking information to the agents. It also implements a defender for the testing of agents in adversarial settings.

ACKNOWLEDGMENTS

The authors acknowledge support by the Strategic Support for the Development of Security Research in the Czech Republic 2019–2025 (IMPAKT 1) program, by the Ministry of the Interior of the Czech Republic under No. VJ02010020 – AI-Dojo: Multi-agent testbed for the research and testing of AI-driven cyber security technologies.

REFERENCES

Andrew, A., Spillard, S., Collyer, J., and Dhir, N. (2022). Developing Optimal Causal Cyber-Defence Agents via Cyber Security Simulation. arXiv:2207.12355.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). OpenAI Gym. arXiv:1606.01540 [cs].

Drašar, M., Moskal, S., Yang, S., and Zat'ko, P. (2020). Session-level Adversary Intent-Driven Cyberattack Simulator. In *2020 IEEE/ACM 24th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, pages 1–9. ISSN: 1550-6525.

Du, Y., Watkins, O., Wang, Z., Colas, C., Darrell, T., Abbeel, P., Gupta, A., and Andreas, J. (2023). Guiding Pretraining in Reinforcement Learning with Large Language Models. In *Proceedings of the 40th International Conference on Machine Learning*, Honolulu, USA.

Elderman, R., J. J. Pater, L., S. Thie, A., M. Drugan, M., and M. Wiering, M. (2017). Adversarial Reinforcement Learning in a Cyber Security Simulation:. In *Proceedings of the 9th International Conference on Agents and Artificial Intelligence*, pages 559–566, Porto, Portugal. SCITEPRESS.

Hammar, K. and Stadler, R. (2020). Finding Effective Security Strategies through Reinforcement Learning and Self-Play. In *2020 16th International Conference on Network and Service Management (CNSM)*, pages 1–9. ISSN: 2165-963X.

Janisch, J., Pevný, T., and Lisý, V. (2023). NASimEmu: Network Attack Simulator & Emulator for Training Agents Generalizing to Novel Scenarios. arXiv:2305.17246.

Microsoft (2021). CyberBattleSim. Microsoft Defender Reasearch Team.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing Atari with Deep Reinforcement Learning. arXiv:1312.5602 [cs].

OpenAI (2023). GPT-4 Technical Report. arXiv:2303.08774 [cs].

Park, J. S., O'Brien, J. C., Cai, C. J., Morris, M. R., Liang, P., and Bernstein, M. S. (2023). Generative Agents: Interactive Simulacra of Human Behavior. arXiv:2304.03442 [cs].

Shinn, N., Cassano, F., Labash, B., Gopinath, A., Narasimhan, K., and Yao, S. (2023). Reflexion: Language Agents with Verbal Reinforcement Learning. arXiv:2303.11366 [cs].

Standen, M., Lucas, M., Bowman, D., Richer, T. J., Kim, J., and Marriott, D. (2021). CybORG: A Gym for the Development of Autonomous Cyber Agents. arXiv:2108.09118.

Wang, G., Xie, Y., Jiang, Y., Mandlekar, A., Xiao, C., Zhu, Y., Fan, L., and Anandkumar, A. (2023a). Voyager: An Open-Ended Embodied Agent with Large Language Models. arXiv:2305.16291 [cs].

Wang, Z., Cai, S., Liu, A., Ma, X., and Liang, Y. (2023b). Describe, Explain, Plan and Select: Interactive Planning with Large Language Models Enables Open-World Multi-Task Agents. arXiv:2302.01560 [cs].

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

Wu, Y., Min, S. Y., Prabhumoye, S., Bisk, Y., Salakhutdinov, R., Azaria, A., Mitchell, T., and Li, Y. (2023). SPRING: GPT-4 Out-performs RL Algorithms by Studying Papers and Reasoning. arXiv:2305.15486.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. (2023). ReAct: Synergizing Reasoning and Acting in Language Models. arXiv:2210.03629 [cs].