Proving the Utility of Large Language Models in Cybersecurity Simulations: A Comprehensive Examination

Stylianos Kampakis
stelios@thetesseractacademy.com

Fabio Rovai
fabio@thetesseractacademy.com

Marcos Charalambides
marcos.charalambides@electiconsulting.com

Theodosis Mourouzis
theodosis.mourouzis@electiconsulting.com

Chris Hicks
c.hicks@turing.ac.uk

January 18, 2025

Abstract

Cyber threats continue to escalate in both frequency and sophistication, necessitating more adaptive and scalable defense strategies. This paper explores how Large Language Models (LLMs) can bolster cybersecurity simulations by automating the creation of synthetic environments and identifying latent vulnerabilities. We employ YAML as a structured representation format for simulating complex network configurations, thereby enabling Large Language Model-driven pipelines to support and improve reinforcement learning (RL) agent training. Comparative studies examine the advantages of LLM-based techniques over classical approaches such as Q-learning, emphasizing increased efficiency, higher adaptability, and enhanced realism in cyberattack simulations. Our findings underscore the transformative potential of integrating LLMs into cybersecurity research, ultimately paving the way for more intelligent and robust cyber-defense systems.

1 Introduction

The cybersecurity landscape has undergone a paradigm shift, wherein threat actors leverage increasingly sophisticated tactics to exploit vulnerabilities in both conventional and

emerging technological infrastructures. Traditional defense mechanisms, largely reliant on static rule sets and signature-based methods, struggle to adapt against zero-day exploits and polymorphic threats [1, 2]. Consequently, there is an urgent need for advanced simulation frameworks that can accurately represent heterogeneous network environments, facilitate continuous adaptation, and support the rapid development of novel defensive strategies.

Cybersecurity simulations offer controlled, replicable environments for research and development, enabling experimentation with attack vectors, defense algorithms, and training methodologies. Among these approaches, reinforcement learning (RL) has gained prominence, particularly through Deep Reinforcement Learning (DRL) methods [3, 4]. However, a major bottleneck lies in the generation of realistic and scalable simulation environments; manual configuration often proves time-consuming, error-prone, and limited in scope.

Recent advances in Large Language Models (LLMs) hold promise for addressing this bottleneck. Contemporary LLMs demonstrate a remarkable capacity for contextual understanding, structured data generation, and direct code synthesis [5–8]. Nevertheless, minimal prior work has explored the application of LLMs to automate the generation of detailed cybersecurity environments. To bridge this gap, the present study proposes an LLM-driven pipeline that uses YAML as a flexible and human-readable configuration scheme, thereby empowering diverse RL agents to train in dynamically generated attack-defense scenarios.

1.1 Challenges with Existing Methods

1. **Manual Labor Intensity:** Building sophisticated simulation environments with multiple subnets, firewalls, and host vulnerabilities is highly resource-intensive. Many existing approaches rely on skilled domain experts, which becomes infeasible for larger-scale or frequently updated simulations.

2. **Limited Adaptability:** As networks evolve (e.g., expansions, reconfigurations, or introduction of novel technologies), existing digital twins must be manually revised to maintain fidelity, incurring continuous overhead [9].

3. **Computational and Maintenance Costs:** Designing high-fidelity simulation frameworks necessitates extensive computational resources [10], hindering the routine testing and training of RL algorithms.

1.2 Proposed Approach and Contributions

This paper posits that Large Language Models (LLMs) can alleviate these constraints by automatically generating YAML-based cybersecurity environments. Specifically, we present:

1. **A Novel LLM-Driven Pipeline:** We detail an approach that leverages prompt engineering and iterative validation to create realistic network topologies, vulnerability distributions, and security policies.

2. **Comparative Analysis with Classical Methods:** We benchmark LLM-generated scenarios against conventional Q-learning agents to quantify improvements in realism, adaptability, and computational efficiency.

3. **Insight into LLM Robustness:** We discuss the inherent limitations of LLMs in producing highly complex configurations and propose strategies such as retrieval-augmented generation and example-based templating.

The remainder of this paper is structured as follows: Section 2 surveys existing literature on machine learning in cybersecurity, digital twin simulation environments, and LLM applications. Section 3 describes our system architecture, agent framework, and evaluation metrics. Section 4 presents the empirical outcomes, and Section 5 discusses the implications of our findings. Finally, Section 6 concludes the paper with potential future directions.

2 Related Work

Research in cybersecurity simulations has been influenced by multiple paradigms, including machine learning-based intrusion detection systems, digital twins for replicating physical systems, and multi-agent frameworks for coordinating attack and defense strategies.

2.1 Machine Learning in Cybersecurity

Machine learning approaches have significantly advanced the identification and mitigation of cyber threats, spanning intrusion detection, malware analysis, and anomaly detection [1, 2]. Deep Reinforcement Learning methods have further demonstrated efficacy in adapting to dynamic adversarial patterns by updating defense strategies in response to novel attack behaviors [3].

2.2 Digital Twins and Simulation Environments

The concept of a digital twin, a virtual replica of physical or cyber-physical systems, has been applied in cybersecurity to enable environment virtualization [11, 12]. While frameworks such as CyberBattleSim [13], YAWNING-TITAN [14], NASim [15], PrimAITE [16], CSLE [17], and NetSecGame [18] offer modular simulation settings, they still require manual or semi-automated processes for scenario generation. This limitation narrows the scope of experimentation and hinders rapid iteration. Additionally, scaling such simulations remains non-trivial due to reconfiguration demands and alignment with real-world network properties.

2.3 LLMs in Cybersecurity

Recent literature explores using LLMs for specialized tasks, including automated exploit generation, detection of zero-day vulnerabilities, and contextual intrusion detection [18–22]. However, few studies have harnessed LLMs for environment creation. The success of LLMs in generating structured data in other domains (e.g., 3D scene generation [5], supply chain optimization [23], and reward design [24]) motivates our investigation into automated cybersecurity environment generation.

2.4 Structured Output Generation with LLMs

Generating accurate, well-formed YAML files exemplifies a structured output task. Several approaches to structured generation exist:

1. *Model Fine-Tuning*: Continually train a foundation model on domain-specific, example-based data to produce more consistent structured outputs [25–27].
2. *Grammar-Constrained Decoding*: Leverage domain-specific languages (DSLs) and grammar-enforcing token generation [28–32].
3. *Schema Engineering*: Use type or interface definitions to guide and validate generation outputs [33].

We adapt a hybrid approach that uses "prompt engineering" and "example-based configuration" to encourage syntactically valid and semantically coherent YAML generation, validated by domain-specific checks.

3 Methodology

In this section, we detail our system design for automated cybersecurity environment generation and the subsequent integration of reinforcement learning (RL) agents. We also describe the metrics used to evaluate the quality of generated environments and agent performance.

3.1 System Architecture

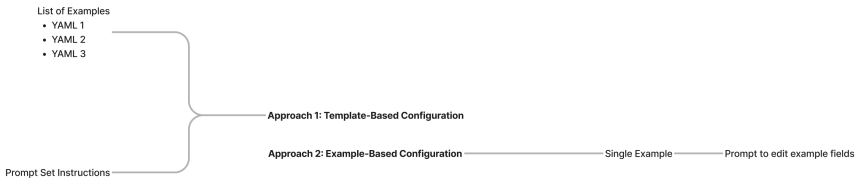

Figure 1: High-level architecture for YAML file creation and experimentation. This image shows how Approach 1 (Template-Based) differs from Approach 2 (Example-Based) for generating YAML configurations.

We develop a pipeline that automatically generates YAML configurations using an LLM and tests them within a cybersecurity emulator (e.g., NASim). Figure 1 illustrates two main approaches for YAML generation:

1. **Template-Based Configuration (Approach 1):** Uses rigid YAML schemas that the LLM fills in according to user prompts.

2. **Example-Based Configuration (Approach 2):** Starts from one or more "golden" YAML files. The LLM edits or extends them in response to user instructions, typically achieving better success rates.

3.1.1 LLM-Driven YAML Generation

After receiving a prompt (including a documentation link or special instructions), the LLM generates a candidate YAML configuration. We apply topological reinforcement checks to ensure coherence and realism, discarding outputs that fail these checks.

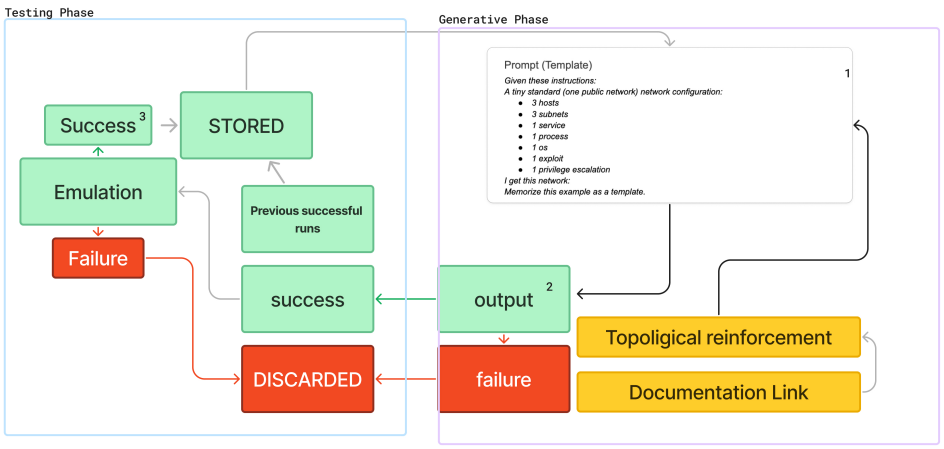

Figure 2: Process flow for YAML creation, including topological reinforcement. If the configuration passes validation and successfully runs in the NASim emulator, it is stored; otherwise, it's discarded.

Figure 2 shows how we validate each YAML file before final acceptance. This includes network adjacency correctness, vulnerability definitions, and firewall rule alignment.

3.2 Reinforcement Learning Integration

After generating a valid YAML file, we instantiate RL-based agents or classical attackers (e.g., Q-learning, PPO, or heuristic scripts) to attempt to compromise the environment. Figure 3 shows a high-level view of how these agents are constructed, given the finalized YAML.

3.3 Overall System Pipeline

Figure 4 synthesizes the entire workflow. By integrating YAML generation and agent testing into a single pipeline, we can rapidly iterate over multiple scenarios, store successful outputs, and feed back any errors into subsequent prompt refinements.

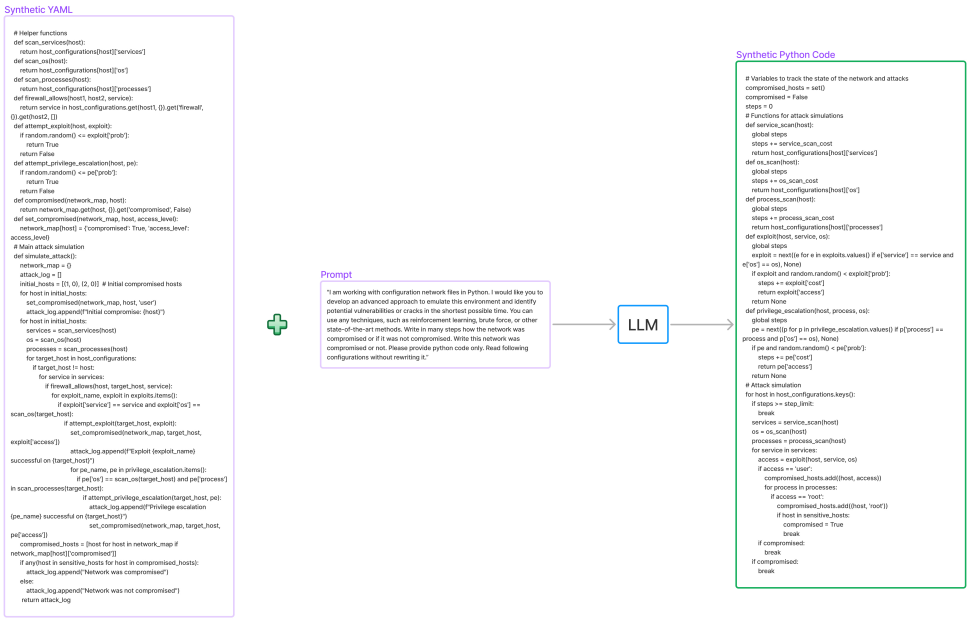

Figure 3: Agent creation process. Once a valid YAML file is confirmed, various RL or heuristic-based agents can be instantiated to emulate attacks and measure time-to-compromise.

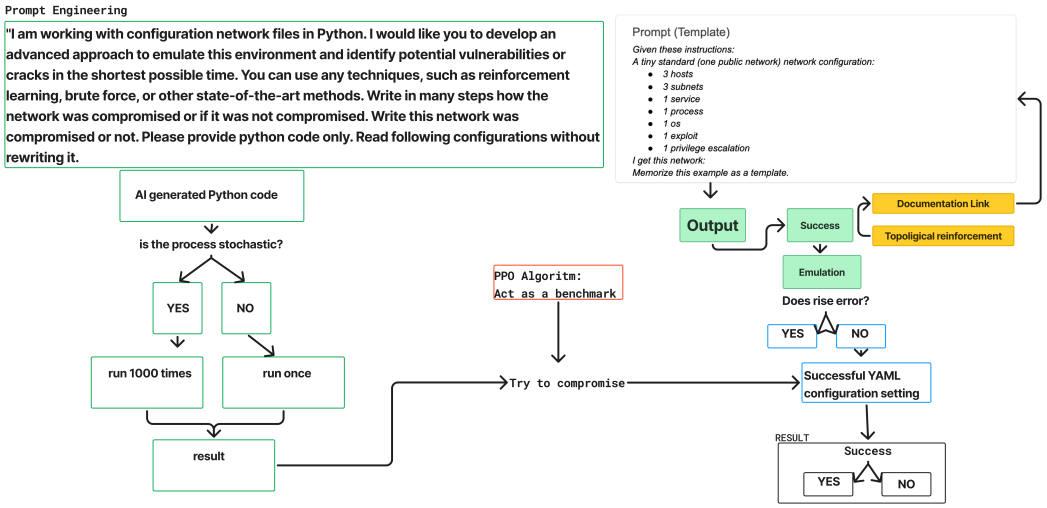

Figure 4: End-to-end workflow of the system, from prompt engineering and LLM-based YAML generation to agent emulation and final outcomes. Successful runs are stored, while failing configurations are refined or discarded.

3.4 Agent Framework

We develop three specialized RL-based agents and also compare them to classical approaches. Each agent is prompted by the LLM under a scenario:

> *"I am working with configuration network files in Python. I would like you to develop an advanced approach to emulate this environment and identify potential vulnerabilities or cracks in the shortest possible time. You can use any techniques, such as reinforcement learning, brute force, or other state-of-the-art methods. Write in many steps how the network was compromised or if it was not compromised. Write 'this network was compromised' or 'not compromised.' Please provide python code only. Read following configurations without rewriting it."*

3.4.1 Agent 1: Probing-Based Approach

- **Focus:** Systematic scanning, enumeration of vulnerabilities, exploitation of the first known open door.
- **Implementation Details:** Typically uses a breadth-first search over IP ranges, testing default credentials or known exploits.
- **Use Case:** Acts as a baseline, providing insight into how quickly straightforward enumeration can compromise a network.

3.4.2 Agent 2: Cyber Kill Chain Simulation

- **Focus:** Simulates multi-stage attacks (reconnaissance, weaponization, delivery, exploit, C2).
- **Implementation Details:** Gains reward each time it successfully completes a stage, penalized for repeated failures at a stage.
- **Use Case:** Represents a more realistic adversary, aligning with Lockheed Martin's cyber kill chain methodology.

3.4.3 Agent 3: Stealthy Privilege Escalation

- **Focus:** Prioritizes lateral movement, zero-trust evasion, and minimal observable footprints.
- **Implementation Details:** Uses partial observability strategies and stealth reward shaping.
- **Use Case:** More advanced APT-style threat models. Allows testing of sophisticated defensive policies.

3.5 Evaluation Metrics

We adopt both *environment generation* and *agent performance* metrics to comprehensively assess our system.

3.5.1 Environment Generation Metrics

- **Success Rate:** Percentage of generated YAML configurations that pass all validation checks.
- **Generation Speed:** Time (in seconds) from user prompt to a valid, simulation-ready environment.
- **Debugging Time:** Effort required to correct or refine invalid files.

3.5.2 Agent Performance Metrics

- **Compromise Rate:** The ratio of successful compromises to total attempts.
- **Time-to-Compromise:** Steps or episodes needed before a host is compromised.
- **Stealth Score (Agent 3 Only):** A weighted penalty for generating alerts or triggering intrusion detection.

3.6 Comparative Benchmarks

We benchmark our three LLM-generated agents against:

- **Approach 0: PPO-Based RL Agent:** A standard Proximal Policy Optimization (PPO) method implemented via Stable-Baselines3. Configured with a multi-layer perceptron (MLP) policy, $\alpha \approx 3 \times 10^{-4}$, $\epsilon = 0.2$, and discount factor $\gamma = 0.99$.
- **Template-Driven Baseline:** Scenarios produced entirely by static templates lacking dynamic vulnerabilities or subnetwork variation.

4 Results and Findings

4.1 YAML Generation Success

Table 1 summarizes the performance of various LLM-based generation methods. The *example-based approach* attains a **70% success rate** in producing valid configurations without manual debugging, outperforming template-only strategies that suffer from frequent adjacency matrix inconsistencies.

4.2 Agent Performance Analysis

We tested each agent on a set of generated network scenarios. Table 2 provides an aggregated view of their performances.

Table 1: LLM-Generated YAML Success Rates (25 runs per method).

Approach	Valid Configs (%)	Common Failure Mode
Template-Based	40%	Syntax errors
Prompt-Based	55%	Firewall rule mismatches
Example-Based	70%	Occasional incomplete adjacency

Table 2: Aggregated Performance Metrics for Generated Agents (averaged over 100 runs).

Agent	Compromise Rate	Avg. Steps	Stealth Score	Notes
Agent 1 (Probing)	100%	12	N/A	Quick to exploit known weaknesses
Agent 2 (Kill Chain)	100%	15	N/A	Realistic multi-stage approach
Agent 3 (Stealth)	85%	28	High	Sometimes fails due to stealth constraints

4.2.1 Probing-Based Agent (Agent 1)

This straightforward strategy enumerates and exploits discovered weaknesses quickly, making it a reliable baseline. It requires fewer interactions, reflecting the simplicity of scanning easily detectible vectors.

4.2.2 Kill Chain Agent (Agent 2)

Aligning with real-world adversarial methodologies, this agent traverses each kill chain phase. Results illustrate a 100% success rate but slightly higher steps compared to Agent 1 due to multi-stage tactics.

4.2.3 Stealth and Privilege Escalation Agent (Agent 3)

Though more advanced, Agent 3 demonstrates lower success rates in some environments, largely attributed to stealth-imposed penalties. It excels in scenarios that reward covert infiltration, but might prolong time-to-compromise.

4.3 Benchmarking Against Approach 0 (PPO-Based RL)

We also compared the LLM-generated agents to a conventional PPO algorithm. While PPO consistently compromised the environment in most runs, it demonstrated longer training and execution times. Its performance advantage lay in its ability to adapt to each environment with minimal prior domain knowledge, though at the cost of computational overhead.

5 Discussion

Our findings confirm that LLMs can efficiently produce valid, reasonably realistic cybersecurity environments, accelerating the testing and training of RL-based agents. Notably, example-based generation methods maintain higher success rates than template-based or

purely prompt-based approaches, reaffirming the value of "golden" reference YAML files in guiding model outputs.

Despite these advancements, challenges persist:

- **Debugging Complex Environments:** As environment complexity increases (e.g., multi-subnet, cross-continental routing), the LLM occasionally generates inconsistent firewall or adjacency rules.
- **Stealth vs. Performance Trade-Off:** Advanced agents designed for low observability face an inherent trade-off between stealth and compromise rate. Balancing these objectives remains an active area of research in multi-objective RL.
- **Generalization Concerns:** While LLMs exhibit powerful generation capabilities, context or domain shifts (e.g., novel OS platforms, custom routing protocols) can degrade environment validity without additional fine-tuning or domain adaptation.

Future work may employ retrieval-augmented generation (RAG) to ground the LLM in a database of validated YAML schemas, improving accuracy and reducing hallucinations. Moreover, large-scale multi-agent scenarios, where attacker and defender agents simultaneously adapt in an evolving environment, represent a promising direction for deepening realism.

6 Conclusion and Future Directions

This paper presents a comprehensive investigation into the utility of Large Language Models for generating realistic cybersecurity environments and training reinforcement learning agents. Our results indicate that LLM-driven pipelines can decrease the manual overhead traditionally associated with digital twin creation, offering a more scalable avenue for both academic research and industry applications.

In summary:

1. **LLM Efficacy:** We demonstrated successful YAML generation with a 70% out-of-the-box validity rate, substantially streamlining environment creation.
2. **Adaptive RL Agents:** LLM-generated adversarial agents (probing, kill chain, stealth) proved effective in multiple scenarios, illustrating robust threat emulation capabilities.
3. **Room for Improvement:** Further work is needed to address debugging complexities, incorporate advanced domain-specific knowledge, and optimize stealth-based reward structures.

Moving forward, we plan to:

- **Enhance Realism:** Integrate real-world logs or network scans into the generation pipeline for even closer alignment with production environments.
- **Expand Multi-Agent Scenarios:** Simultaneously evolve attackers and defenders for more nuanced and dynamic RL training experiences.

- **Explore Fine-Tuning:** Investigate domain-specific fine-tuning of LLMs to yield higher success rates and improved fidelity in YAML generation.

Acknowledgments

We gratefully acknowledge the Alan Turing Institute and the Department for AI for CyberDefense for providing the financial support that made this research possible.

Acknowledgments

We gratefully acknowledge the support of colleagues and domain experts who provided insights and feedback on the architecture. Their guidance significantly contributed to the experimentation and analysis presented in this paper.

References

[1] A. Handa, A. Sharma, and S. K. Shukla, "Machine learning in cybersecurity: A review," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 9, no. 4, p. e1306, 2019.

[2] D. Dasgupta, Z. Akhtar, and S. Sen, "Machine learning in cybersecurity: a comprehensive survey," *The Journal of Defense Modeling and Simulation*, vol. 19, no. 1, pp. 57–106, 2022.

[3] T. T. Nguyen and V. J. Reddi, "Deep reinforcement learning for cyber security," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 34, no. 8, pp. 3779–3795, 2021.

[4] Limmen, "Awesome RL for Cybersecurity," GitHub, 2024. [Online]. Available: `https://github.com/Limmen/awesome-rl-for-cybersecurity`

[5] R. Ma, A. G. Patil, M. Fisher, M. Li, S. Pirk, B.-S. Hua, S.-K. Yeung, X. Tong, L. Guibas, and H. Zhang, "Language-driven synthesis of 3D scenes from scene databases," *ACM Transactions on Graphics (TOG)*, vol. 37, no. 6, p. 1–16, 2018.

[6] A. X. Chang, M. Eric, M. Savva, and C. D. Manning, "SceneSeer: 3D scene design with natural language," *arXiv preprint arXiv:1703.00050*, 2017.

[7] S. Long, T. Schuster, and A. Piché, "Can large language models build causal graphs?," *arXiv preprint arXiv:2303.05279*, 2023.

[8] T. Jiralerspong, X. Chen, Y. More, V. Shah, and Y. Bengio, "Efficient causal graph discovery using large language models," *arXiv preprint arXiv:2402.01207*, 2024.

[9] Y. Wu, K. Zhang, and Y. Zhang, "Digital twin networks: A survey," *IEEE Internet of Things Journal*, vol. 8, no. 18, pp. 13789–13804, 2021.

[10] M. Segovia and J. Garcia-Alfaro, "Design, modeling and implementation of digital twins," *Sensors*, vol. 22, no. 14, p. 5396, 2022.

[11] R. Faleiro, L. Pan, S. R. Pokhrel, and R. Doss, "Digital twin for cybersecurity: Towards enhancing cyber resilience," in *Broadband Communications, Networks, and Systems*, Springer, 2022, pp. 57–76.

[12] M. Eckhart and A. Ekelhart, "Towards security-aware virtual environments for digital twins," in *Proceedings of the 4th ACM Workshop on Cyber-Physical System Security*, 2018, pp. 61–72.

[13] Microsoft, *CyberBattleSim*, 2021. [Online]. Available: `https://github.com/microsoft/CyberBattleSim`

[14] Defence Science and Technology Laboratory, *YAWNING-TITAN: A Cyber Security Research Platform*, 2024. [Online]. Available: `https://github.com/dstl/YAWNING-TITAN`

[15] J. Schwartz, *NetworkAttackSimulator: A simple OpenAI gym environment for simulating network attacks*, 2024. [Online]. Available: `https://github.com/Jjschwartz/NetworkAttackSimulator`

[16] Autonomous-Resilient-Cyber-Defence, *PrimAITE*, 2023. [Online]. Available: `https://github.com/Autonomous-Resilient-Cyber-Defence/PrimAITE`

[17] K. Hammar, *CSLE: A platform for evaluating and developing reinforcement learning agents for control problems in cyber security*, 2023. [Online]. Available: `https://github.com/Limmen/csle`

[18] M. Rigaki, O. Lukáš, C. A. Catania, and S. Garcia, "Out of the cage: How stochastic parrots win in cyber security environments," *arXiv preprint arXiv:2308.12086*, 2023.

[19] W. Tann, Y. Liu, J. H. Sim, C. M. Seah, and E.-C. Chang, "Using large language models for cybersecurity capture-the-flag challenges and certification questions," *arXiv preprint arXiv:2308.10443*, 2023.

[20] M. A. Ferrag, M. Ndhlovu, N. Tihanyi, L. C. Cordeiro, M. Debbah, and T. Lestable, "Revolutionizing cyber threat detection with large language models," *arXiv preprint arXiv:2306.14263*, 2023.

[21] M. D. Purba, A. Ghosh, B. J. Radford, and B. Chu, "Software vulnerability detection using large language models," in *2023 IEEE 34th International Symposium on Software Reliability Engineering Workshops (ISSREW)*, IEEE, 2023, pp. 112–119.

[22] M. M. Yamin, E. Hashmi, M. Ullah, and B. Katt, "Applications of llms for generating cyber security exercise scenarios," 2024.

[23] B. Li, K. Mellou, B. Zhang, J. Pathuri, and I. Menache, "Large language models for supply chain optimization," *arXiv preprint arXiv:2307.03875*, 2023.

[24] Y. J. Ma, W. Liang, G. Wang, D. Huang, O. Bastani, D. Jayaraman, Y. Zhu, L. Fan, and A. Anandkumar, "Eureka: Human-level reward design via coding large language models," *arXiv preprint arXiv:2310.12931*, 2023.

[25] B. Zhang, Z. Liu, C. Cherry, and O. Firat, "When scaling meets llm finetuning: The effect of data, model and finetuning method," *arXiv preprint arXiv:2402.17193*, 2024.

[26] C. Jeong, "Fine-tuning and utilization methods of domain-specific llms," *arXiv preprint arXiv:2401.02981*, 2024.

[27] J. Wei, M. Bosma, V. Y. Zhao, K. Guu, A. W. Yu, B. Lester, N. Du, A. M. Dai, and Q. V. Le, "Finetuned language models are zero-shot learners," *arXiv preprint arXiv:2109.01652*, 2021.

[28] T. Lin, "Generating Structured Output from LLMs," `https://www.timlrx.com/blog/generating-structured-output-from-llms`, accessed 2024-11-27.

[29] Microsoft, *guidance*, 2023. [Online]. Available: `https://github.com/guidance-ai/guidance`

[30] SRI Lab, ETH Zurich, *LMQL*, 2023. [Online]. Available: `https://github.com/eth-sri/lmql`

[31] dottxt-ai, *outlines*, 2024. [Online]. Available: `https://github.com/dottxt-ai/outlines`

[32] ggerganov, *llama.cpp*, 2023. [Online]. Available: `https://github.com/ggerganov/llama.cpp`

[33] Microsoft, *TypeChat*, 2023. [Online]. Available: `https://github.com/microsoft/TypeChat`