Journal of
Cyber Security

Tech Science Press

https://doi.org/10.32604/jcs.2026.075976

ARTICLE

GenAI-Powered Autonomous Cyber Offense-Defense: An Explainable LLM Red-vs-Blue Simulation and Self-Defense Framework

Haitian Du*

School of Information Engineering, Henan University of Animal Husbandry and Economy, Zhengzhou, China

*Corresponding Author: Haitian Du. Email: 17516465123@163.com

Received: 12 November 2025; Accepted: 19 January 2026; Published: 25 May 2026

ABSTRACT: Modern cyberattacks evolve rapidly, overwhelming static and rule-based defenses. This paper proposes GenAI-Powered Autonomous Cyber Offense-Defense, a closed-loop framework in which large language models (LLMs) control both a red-team attacker and a blue-team defender. The agents operate in a simulated enterprise network, generate natural-language rationales for every action, and update defensive policies through a self-adaptive learning loop. We instantiate the framework with LLM-based agents that plan multi-stage attacks, detect anomalies, and autonomously execute containment and hardening actions. In experiments on a three-host virtualized testbed and a scalable multi-node emulation, the adaptive blue agent reduces the attacker's success rate from 72% to 5% over six iterations and cuts mean detection latency from 5.4 to 1.2 s compared with a non-learning baseline. Explainability experiments with security analysts show that increasing explanation completeness from 0.75 to 0.95 raises subjective trust scores by roughly 46% and improves decision alignment with experts to about 90%. These results demonstrate that GenAI can orchestrate realistic attack–defense exercises and produce self-improving cyber defenses that remain transparent to humans. The proposed framework offers a reusable platform for evaluating LLM-based security agents and studying AI-on-AI red–blue co-evolution in cybersecurity.

KEYWORDS: Generative artificial intelligence; large language models; autonomous red teaming; autonomous cyber defense; explainable AI; attack-defense simulation

1 Introduction

The rapid advancement of generative artificial intelligence (GenAI) is reshaping the cybersecurity landscape for both attackers and defenders. Large Language Models (LLMs) capable of sophisticated reasoning and code generation have emerged as powerful tools in this domain. Adversaries are increasingly leveraging LLMs to automate cyberattacks—from crafting highly convincing phishing campaigns to generating exploit code and malware—thereby lowering the barrier for launching complex attacks at scale [1]. Conversely, defenders are exploring AI-driven solutions for threat detection, incident response, and vulnerability management. This dual-use of GenAI in cyber offense and defense has led to an emerging "arms race" in which AI agents compete on both sides of the security equation.

Traditionally, organizations have relied on red team vs. blue team exercises to assess and strengthen their security posture. In such exercises, human red teams emulate attackers, probing systems for weaknesses, while blue teams attempt to detect and mitigate the attacks in real-time. These engagements provide valuable insights but are typically labor-intensive and periodic, leaving potential blind spots between exercises. Moreover, as attack techniques evolve rapidly (often aided by AI), purely manual defense strategies struggle

Copyright © 2026 The Author. Published by Tech Science Press. This work is licensed under a Creative Commons Attribution 4.0 International License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

to keep pace. This motivates the need for an autonomous, continuous red-vs.-blue simulation framework, where AI agents can relentlessly test and fortify systems at machine speed.

In this paper, we propose GenAI-Powered Autonomous Cyber Offense-Defense, a novel framework that harnesses LLMs to simulate both cyber attackers and defenders in a closed-loop environment. Our approach features an LLM-driven Red Team agent that can plan and execute attacks (e.g., network intrusion, exploitation, social engineering) and an LLM-driven Blue Team agent that monitors, detects, and responds to these attacks. Crucially, our framework emphasizes explainability: each agent's decisions are accompanied by human-readable explanations of their reasoning and tactics. By integrating explainable AI (XAI) techniques, we ensure that the automated strategies remain transparent to security analysts, fostering trust and enabling users to interpret the simulated engagement.

XAI Trace Data. To operationalize explainability beyond narrative descriptions, we log an XAI trace at every simulation step. Each trace stores (i) the agent observation/state summary, (ii) the structured action in JSON, (iii) the natural-language rationale, and (iv) the environment outcome (success/failure, alerts) with timestamps and agent role (Red/Blue). These traces constitute the XAI dataset used in Sections 3.4 and 4.4 to quantify explanation completeness, human trust, debugging utility, and decision alignment.

The system also includes a self-adaptive learning loop for the defensive (blue) agent, allowing it to improve its countermeasures over successive attack iterations. In essence, the red and blue agents engage in an ongoing duel, and the blue agent "learns" from each attack to better defend against future ones—a form of autonomous self-defense optimization [2]. Unlike recent LLM-based SOC copilots and incident-response assistants, which primarily support human analysts, our framework uses LLMs as fully autonomous red and blue agents that interact within a closed simulation loop.

The proposed framework is platform-agnostic and research-oriented. We design a plausible simulation environment where the LLM agents interact with a virtual network, mimic real-world tactics, techniques, and procedures (TTPs), and adapt to new scenarios without human intervention. The goal is to enable continuous, AI-driven security assessments that can reveal vulnerabilities and test defenses far more frequently and thoroughly than human teams alone. Additionally, by capturing the rationale behind each agent's actions, the framework produces an explainable record of the offensive and defensive maneuvers. This not only aids in validating AI behavior but also provides educational insights—for example, helping analysts understand how a particular exploit succeeded or why a defensive measure failed.

Contributions: The key contributions of this work are as follows:

Autonomous Red-vs.-Blue Simulation: We design a closed-loop system in which LLM-based agents emulate a red team and a blue team that continuously attack and defend a simulated enterprise network without human intervention.

Explainable Offense–Defense Orchestration: Both agents output structured actions together with natural-language rationales, enabling fine-grained inspection of attack chains and defensive responses and providing a basis for standardized XAI evaluation. We additionally curate an XAI evaluation set by sampling logged traces and attaching human ratings and expert-alignment labels, enabling reproducible measurement of explanation quality and trust.

Self-Adaptive Defense Mechanism: We introduce a lightweight learning loop that turns failed defenses into prompt and rule updates for the blue agent, allowing it to harden its policy over iterations and co-evolve with the red agent.

Comprehensive Empirical Study: On a realistic three-host testbed, we quantify how the GenAI blue agent reduces attack success probability, shortens detection latency, and increases human trust in explanations compared with a scripted baseline.

The remainder of this paper is organized as follows. Section 2 reviews related work in AI-driven cyber offense and defense, red teaming methodologies, and explainable AI in security. Section 3 details our framework's design and methodology, including the architecture of the LLM agents, the explainability components, and the adaptive learning mechanism. Section 4 presents the experimental setup and results of our red-vs.-blue simulations, with analysis comparing the performance of our GenAI agents to baseline strategies. Finally, Section 5 concludes the paper, discussing key findings, security implications, and directions for future research in autonomous AI-driven cybersecurity.

2 Related Work

AI-Driven Cyber Offense: A growing body of research and development has explored the use of artificial intelligence to augment offensive cyber capabilities. Attackers can leverage AI and machine learning to automate tasks that traditionally require human expertise. For instance, machine learning models have been used to crack passwords (e.g., training generative models like GANs to produce probable passwords), to craft spear-phishing emails that bypass spam filters by mimicking human writing, and to generate malicious URLs or malware that evade detection. Notable examples include PassGAN for password guessing and DeepPhish for phishing URL generation, which learn patterns from real credentials and phishing sites to create more effective attacks. AI has also been applied to vulnerability discovery and exploit development—an AI system can scan software for potential flaws or even generate exploit code for known vulnerabilities. Early prototypes like DeepExploit integrated reinforcement learning with penetration testing frameworks to autonomously choose exploits against target systems. Similarly, IBM's DeepLocker project demonstrated how AI could hide a malicious payload within benign software and trigger it only under specific conditions (using facial recognition as the trigger), illustrating the concept of intelligent, stealthy malware. These advancements indicate that well-resourced adversaries could increasingly deploy AI to increase the speed, scale, and sophistication of attacks. However, most AI-driven offense efforts to date have focused on specific attack vectors or tools rather than a comprehensive autonomous attacker with general reasoning ability. Moreover, such offensive AI agents are typically not used in an interactive red-team exercise against a live defender agent; instead, they often assume a static target or scenario.

AI-Driven Cyber Defense [3]: On the defensive side, AI techniques have been widely studied and applied for threat detection and response. Anomaly detection systems using machine learning can identify deviations in network traffic or user behavior that may indicate intrusions, often with greater adaptability than rule-based systems. For example, various deep learning models have been employed to detect malware or network intrusions by learning complex patterns in data that signify malicious activity. Security information and event management (SIEM) tools have started incorporating AI to triage alerts and reduce false positives. Furthermore, recent work has looked at using LLMs to assist defense in tasks such as log analysis, summarizing threat intelligence reports, and even generating human-readable incident reports. One emerging direction is the use of LLMs as decision-making engines in automated incident response: given a description of a security incident (affected systems, indicators of compromise), an LLM can recommend or execute appropriate response actions (such as isolating a host, applying a patch, or updating firewall rules) based on its learned knowledge of best practices. Although research on LLM-powered defense agents is in its infancy, preliminary studies suggest that LLMs can encode a wealth of cybersecurity knowledge and could serve as adaptive playbook generators for defenders. Prior to LLMs, reinforcement learning agents were investigated for autonomous cyber defense—for instance, agents trained to dynamically reconfigure network defenses or deceive attackers (moving target defenses, honeypots, etc.) in response to attacks. The U.S. DARPA Cyber Grand Challenge (2016) provided an early glimpse of autonomous cyber defense and offense: it was a competition where fully automated systems had to find vulnerabilities in software, exploit

them, and patch them on the fly, effectively acting as both red and blue teams. The winning system, Mayhem, demonstrated that automation could handle certain software vulnerability battles faster than humans, though it operated on isolated CTF-style challenges. Our work builds on this vision of automation but extends it to more general scenarios and leverages the generative and reasoning capabilities of LLMs for both offense and defense roles.

Red-vs.-Blue Simulation and Autonomous Agents: There have been efforts to create simulated environments for adversarial engagements in order to train or evaluate autonomous agents. Notably, Microsoft's CyberBattleSim is an open-source simulation where a red agent and a blue agent compete in a model network; it uses a reinforcement learning framework (OpenAI Gym) to train agents to perform actions like lateral movement or intrusion detection. Extensions of this work and similar platforms (e.g., CybORG, GyoiThon) have shown that multi-agent reinforcement learning can produce competent offensive and defensive strategies under simplified conditions. Joint training of red and blue agents in these environments can lead to an arms race dynamic where each agent improves in response to the other, echoing game-theoretic principles. However, RL-based agents typically require a large number of training episodes, and their learned policies are hard to interpret. In practical terms, purely autonomous red-team tools exist (sometimes called automated penetration testing or breach-and-attack simulation products), which can continuously test an environment's defenses using scripted techniques or AI planning. Likewise, on the defense side, there are automated intrusion prevention systems that can take certain actions (like blocking an IP or quarantining a file) without human intervention once a threat is detected. Yet, these systems are usually based on predefined rules or learned patterns in specific domains—they do not have the broad situational reasoning that an LLM offers, nor do they typically generate explanations for their actions. Our framework differentiates itself by using an LLM-based approach to drive the decision-making for both red and blue agents, allowing richer strategic reasoning (through natural language planning and knowledge) and flexibility in the range of scenarios. Moreover, by having the two sides interact in a loop, we simulate a continuous engagement akin to "purple teaming" (collaborative offense-defense) but in an automated way, enabling continuous security assessment [4].

Explainable AI in Cybersecurity: The importance of explainability in AI-driven security solutions has been widely acknowledged. Cybersecurity operations are high-stakes and require trust in automated tools; analysts need to understand why a detection was flagged or why a certain response was taken. Traditional machine learning models like neural networks are often black boxes, making their outputs hard to justify. Thus, researchers have pursued explainable AI (XAI) techniques in this domain, such as decision trees or rule extraction for malware classification, feature attribution methods for intrusion detection (to highlight which network features contributed most to an alert), and natural language explanations generated for security recommendations. Some studies show that providing explanations for AI outputs can improve analysts' confidence and effectiveness—for example, an explainable intrusion detection system might accompany an alert with a sentence like "This login was flagged because it deviated from the user's usual location and occurred outside normal working hours," which is more actionable than a raw anomaly score [5]. Despite these advances, most existing security AI tools do not yet have robust explainability built-in; it's an active area of research to balance model complexity with interpretability. In the context of autonomous red and blue agents, explainability is even less explored. While an RL agent or a scripted attack simulator might achieve its goal, it's often unclear how or why it chose a certain path (for instance, which vulnerability it exploited first or what indicators led a defense agent to isolate a host). Our work aims to fill this gap by design: by leveraging LLMs, which can inherently produce text explanations, we integrate XAI at the core of the red-vs.-blue simulation. This means our red agent can articulate its thought process (e.g., "I chose to exploit the web server because its software version is outdated and likely vulnerable") and the blue agent can explain

its decisions ("I am blocking traffic from IP X because I observed a port scan from that address"). This approach draws inspiration from prior work on explainable agents and aligns with calls in the cybersecurity community for greater transparency in AI-driven tools.

In summary, while there is prior work in using AI for cyber offense, AI for cyber defense, autonomous red-teaming simulations, and explainable AI in security, our work is novel in unifying these threads. We deliver an autonomous red-vs.-blue framework powered by LLMs that not only engages in a realistic cyber battle but also provides interpretable insights into each agent's behavior. This combination of capabilities addresses a unique intersection of challenges not fully met by earlier research, providing a platform to explore how generative AI can be safely and effectively applied in adversarial cybersecurity scenarios.

2.1 AI-Driven Cyber Offense

Recent years have witnessed growing interest in automated red teaming and attack simulation, aiming to reduce the cost and human effort required for security assessment. Systems such as AutoPentest and related commercial breach-and-attack simulation platforms automate predefined attack playbooks to evaluate defensive coverage, but they largely rely on scripted logic and static decision trees, limiting their adaptability to novel environments.

Academic research has explored more flexible approaches. CyberBattleSim models cyber offense and defense as a reinforcement learning problem, where an attacker agent learns to traverse a network graph and exploit vulnerabilities through trial-and-error interactions. While effective for studying attack path optimization, these RL-based attackers typically require extensive training episodes and provide limited interpretability regarding why specific attack decisions are chosen. Similarly, MalGEN and related multi-agent frameworks focus on generating coordinated malicious artifacts or attack behaviors, emphasizing attack realism and diversity rather than continuous interaction with an adaptive defender.

In contrast, our work differs from prior automated red-team approaches in three key aspects. First, we leverage large language models (LLMs) as general-purpose reasoning engines rather than relying on scripted playbooks or narrowly trained RL policies. Second, the Red Team agent operates in a closed-loop adversarial setting, continuously reacting to Blue Team responses rather than attacking a static environment. Third, every offensive action is accompanied by a natural-language explanation, enabling analysts to inspect not only what attack was performed, but also the attacker's strategic rationale—an aspect largely absent in prior automated red-team systems.

2.2 AI-Driven Cyber Defense

On the defensive side, artificial intelligence has long been applied to intrusion detection, anomaly detection, and automated incident response. Traditional machine-learning-based systems focus on classification or anomaly scoring, often acting as black boxes that generate alerts without explicit reasoning. More recently, the emergence of large language models has enabled new forms of LLM-assisted security operations, including log summarization, alert triage, and decision support for security analysts.

A representative example is CyberSleuth, an LLM-powered autonomous SOC agent designed for network intrusion forensics. CyberSleuth integrates tool invocation and stepwise reasoning to analyze network traces and logs, automatically identifying exploited services, vulnerability identifiers, and attack outcomes. This line of work demonstrates the potential of LLMs to automate complex defensive analysis tasks that traditionally require expert human analysts.

However, existing LLM-based defense systems primarily operate in a reactive and post-incident manner, focusing on investigation and reporting after suspicious activity has already been detected. In contrast, our

framework positions the Blue Team LLM as an active, decision-making defender embedded directly in an ongoing red–blue confrontation loop. The Blue agent not only analyzes alerts, but also autonomously executes containment, mitigation, and hardening actions in real time, while continuously adapting its defensive strategy across multiple attack iterations. Moreover, our design explicitly couples defensive decisions with explainable rationales, allowing human operators to understand and audit why specific responses were chosen.

2.3 Explainable AI in Cybersecurity

Explainability has become a critical requirement for AI systems deployed in cybersecurity, where decisions often have high operational and legal impact. Prior research on explainable AI (XAI) in security has focused on making intrusion detection models more transparent, for example, by attributing alerts to influential features in network traffic or providing rule-based explanations for anomaly scores. Such approaches have been shown to improve analyst trust and facilitate debugging of detection models.

In the context of security operations centers (SOCs), XAI techniques are also increasingly studied from perspectives of human trust, regulatory compliance, and operational accountability. Explanations help analysts assess whether an alert is credible, justify response actions during audits, and comply with organizational or legal requirements that demand transparency in automated decision-making. Nevertheless, most existing XAI work in cybersecurity treats explanations as post-hoc artifacts, generated after a model produces a detection result.

Our work departs from this paradigm by integrating explainability as a first-class design principle in an autonomous red–blue system. Rather than explaining isolated predictions, we log explanations jointly with observations, actions, and outcomes at every decision step. This enables systematic evaluation of explanation completeness, trustworthiness, and alignment with expert judgment in a dynamic adversarial setting. By embedding XAI directly into both offensive and defensive agents, our framework extends prior XAI research from static detection tasks to interactive, adaptive cyber operations.

3 Methodology

3.1 System Architecture and Workflow

Our autonomous red-vs.-blue framework consists of three main components: the Red Team agent (offensive LLM), the Blue Team agent (defensive LLM), and the simulated environment that represents the target system/network. The overall architecture is designed as a closed-loop simulation where the red and blue agents interact through the environment in a turn-based fashion. A central controller orchestrates this loop, feeding observations to the agents and applying their actions to the environment state.

The LLM layer is backend-agnostic: agents can be instantiated with different model sizes (local open-source checkpoints or cloud-hosted LLMs). For larger models, inference is served from a dedicated LLM node, while the controller maintains the same action schema to ensure comparability.

Fig. 1 conceptually illustrates this setup (in this text, we describe the figure in words):

To make the architecture more concrete from a systems perspective, the framework can be viewed as four interacting modules: (i) an environment engine that maintains the discrete network state and implements the effects of atomic actions; (ii) a controller that converts environment states into textual observations and parses the structured actions returned by the LLMs; (iii) the Red and Blue LLM agents, each queried via a well-defined JSON-based API returning ⟨action, explanation⟩tuples; and (iv) a logging and metrics collector that records states, actions, and rationales for offline analysis. At every time step, the controller queries the red agent with observation o_t^R, applies the attack action a_t^R, updates the environment

to state s_{t+1}, generates a defender observation o_t^R, and then queries the blue agent for a response a_t^B. This loop continues until an absorbing terminal condition is reached.

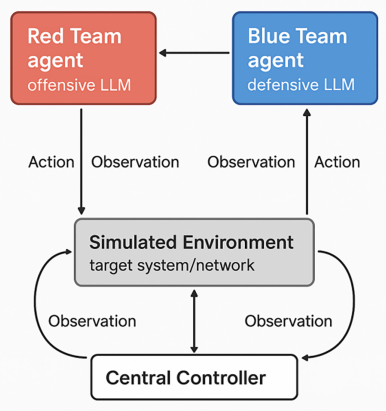

Figure 1: Architecture of the GenAI-powered autonomous Red-vs-Blue simulation framework.

The innovation of this architecture lies in:

1. incorporating generative artificial intelligence (GenAI) into red-blue confrontation games to achieve a self-explanatory, evolvable, and automated attack-defense cycle;
2. ensuring sustainable simulation through a centralized control loop, thereby forming a closed-loop system of self-learning and dynamic defense [6];
3. supporting Explainable AI (XAI) to enhance the transparency and auditability of model behavior.

Red Team LLM Agent: Receives information about the target environment (e.g., network topology, any discovered vulnerabilities or credentials) and outputs an attack action along with its rationale.

Blue Team LLM Agent: Receives information about the system's status and any alerts or signs of compromise, and outputs a defensive action and its rationale.

Environment Model: Encodes the state of the network (hosts, services, vulnerabilities, user accounts, etc.) and the security monitoring systems. It determines the outcomes of actions (e.g., whether an exploit succeeds, or whether an intrusion is detected) based on predefined rules and random factors, and it generates observations for the agents.

Simulation Loop: The interaction proceeds in iterative rounds. In each round, the Red agent acts first (attempting some malicious action) and the Blue agent responds. The sequence in one iteration is as follows:

1. Red Observation: The Red LLM agent is presented with the current state of what it knows about the target. For example: "Host A is observed with ports 80 and 22 open; Host B is not yet discovered; possible credentials for user X obtained from a previous step." This observation may include results of prior reconnaissance steps or any partial knowledge the Red agent has accumulated.

2. Red Action Decision: The Red agent uses the LLM to decide an attack step. This could be a reconnaissance action (scanning a host, enumerating users, etc.), an exploit attempt on a discovered vulnerability, a lateral movement from an already compromised machine, or a strategic move like establishing persistence or exfiltrating data. The agent generates an Action (in a structured format or natural language) describing what it will do, and an Explanation justifying why this action is logical given its goals.

3. Environment Update (Attack): The environment module processes the Red agent's action. It checks if the action is valid and then determines the outcome. For example, if the Red agent decided to exploit Host

A via a buffer overflow (and Host A is indeed vulnerable to that exploit), the environment would mark Host A as compromised and possibly generate an alert (e.g., the host's intrusion detection system triggers an alarm). If the attack fails (say Host A is not vulnerable or the exploit was mis-targeted), the environment might still produce some observable clue (like an error log or a minor network anomaly) or nothing if the action was completely stealthy [7].

4. Blue Observation: The Blue LLM agent is then presented with its view of the world. This includes any security alerts or anomalous events that were triggered by the Red agent's action, as well as the overall system status (which hosts are up, which services are running, any integrity checks failing, etc.). The Blue agent does not have perfect knowledge of what the Red did; it only sees what the monitoring systems would detect. For instance, it might receive an alert "Host A: unusual process execution detected" or see network logs indicating a port scan from a certain IP address. All such cues form the Blue agent's observation.

5. Blue Action Decision: Given this observation, the Blue agent uses the LLM to decide on a defensive response. This could be a containment action (isolating Host A from the network), a mitigative action (applying a patch or shutting down a vulnerable service, blocking an IP at the firewall), an investigative action (initiating a forensic scan on Host A, or checking other hosts for similar activity), or an alert escalation (e.g., informing that a breach is suspected). The Blue agent produces an Action and an Explanation (e.g., "Action: isolate Host A; Explanation: Host A is likely compromised as indicated by X, isolating it will prevent lateral movement.").

6. Environment Update (Defense): The environment processes the Blue agent's action and updates the state. For example, if Host A is isolated, the environment will ensure that the Red agent cannot use Host A to reach other systems (cutting off that path). If the Blue action was to patch a service or change a firewall rule, those changes reflect in the environment (the vulnerability might be removed, or traffic from the attacker's IP is now blocked). The environment also records whether the Blue's action succeeded in mitigating the threat (e.g., containment of Host A stops the attack progression).

Iteration & Termination: The next round begins with the Red agent observing the updated state (which now might include, say, that Host A is unresponsive due to isolation, forcing the attacker to adapt). The cycle repeats, with the Red agent formulating its next move, followed by Blue's response. This continues until an end condition is reached. We define end conditions such as: the Red agent achieves its objective (e.g., compromising a critical server or exfiltrating sensitive data), the Blue agent fully contains/neutralizes the threat (e.g., all attacker access is cut off and systems secured), or a maximum number of rounds elapses (simulating that the attacker gives up or withdraws).

Analysis:

Table 1 defines the full-process indicator system of red-blue confrontation from the perspective of system interactions.

1. Each stage includes input information, decision-making actions, system state updates, and corresponding metrics.
2. 'Attack decision latency' and 'Response latency' reflect the real-time reasoning performance of LLMs;
3. 'Knowledge completeness' and 'Signal-to-noise ratio' measure the informational integrity and the quality of alerts in the simulation environment.
4. "Adaptive efficiency" is a key metric for assessing a system's learning capability, used to evaluate the rate at which defensive efficiency improves after multiple rounds of interaction [8].

These metrics not only support performance evaluation but can also serve as feedback signals for future reinforcement learning or policy optimization algorithms.

Table 1: Simulation loop process metrics.

Iteration Phase	Input Data	Agent Decision Type	Generated Outputs	Environment State Update	Performance Indicators
Red Observation	Host/service map, known credentials, prior exploit results	Threat reasoning & plan synthesis (LLM)	Attack hypothesis (e.g., "Exploit Host A via port 80")	No change (informational phase)	*Knowledge completeness* (%)—ratio of discovered vs. total hosts
Red Action Decision	Observation summary, goal prompt	Action selection + Explanation generation	Action tuple: ⟨Action, Explanation⟩	Possible system compromise	*Attack decision latency* (s); *LLM coherence score*
Environment Update (Attack)	Red Action	Attack validity check, probabilistic outcome computation	Updated environment vector (e.g., compromised host flag)	System state transition (compromise or alert)	*Exploit success rate* (%); *Detection triggered (binary)*
Blue Observation	Security alerts, logs, anomaly scores	Threat inference reasoning (LLM)	Structured event summary	No change (monitoring phase)	*Signal-to-noise ratio* (true alerts/total alerts)
Blue Action Decision	Alerts, observed anomalies	Response strategy formulation (LLM)	Action tuple: ⟨Action, Explanation⟩	Network isolation, patch, or block event	*Response latency* (s); *Rationale interpretability score*
Environment Update (Defense)	Blue Action	Defense validation, patch propagation	Updated defense posture	Risk state recalculation	*Mitigation success rate* (%)
Iteration Termination	Global state vector	End condition check	Outcome label (win/loss)	Final environment snapshot	*Convergence rounds* (#); *Adaptive efficiency (Δsuccess rate)*

Analysis

Table 2 provides a typical multi-round red-blue game data sample, reflecting the evolutionary trends of the system:

1. In the previous rounds, the Red team had a higher probability of successful attacks (0.85), while the Blue team exhibited a delayed response.

2. As the Blue Team enhances its detection strategy through an adaptive learning loop, the detection time is reduced from three steps to one step.
3. "Mitigation Effectiveness" increased from 0.10 to 1.00, indicating a significant improvement in the success rate of defenses.
4. The "Explanation Quality" has consistently maintained a high standard, indicating that the explanations generated by the LLM remain coherent and readable even in multi-turn adversarial scenarios.

Table 2: Red–Blue interaction outcomes (Example scenario results).

Round	Red Team Action	Blue Team Response	Outcome	Detection Time (steps)	Attack Success Probability	Blue Mitigation Effectiveness	Explanation Quality (Score/5)
1	Exploit Host A (Apache RCE)	No response (undetected)	Host A compromised	—	0.85	0.1	4.5
2	Lateral move to Host B	Host A isolation triggered	Partial containment	3	0.45	0.6	4.2
3	Credential brute-force Host C	Password reset & firewall block	Attack blocked	2	0.2	0.85	4.8
4	Phishing attempt on user C	Suspicious email flagged	Preempted (no impact)	1	0.05	0.95	4.9
5	Retry exploit Host A (variant)	Patch applied proactively	Prevented	1	0	1	5

Implementation and Reproducibility. The current prototype is implemented in Python 3.11 with a discrete-event simulator for the three-host enterprise network. Red and blue actions are represented as JSON objects and executed through a thin adapter layer that interacts with the environment engine. LLM queries are issued via a stateless API, and all prompts, hyperparameters, and environment configuration files are stored as version-controlled artifacts.

The vertical comparison in this table illustrates the system's self-evolution characteristics (Blue learning → Red suppression), while the horizontal combination of metrics (success rate, detection latency, explanation quality) reflects the co-evolution trajectory of the attack-defense agents.

This loop constitutes a single scenario simulation. Throughout the process, both agents are effectively engaging in a turn-based strategic game, mediated by the environment. The architecture's modularity allows us to plug in different environment scenarios (varying network configurations or attack goals) and to swap or upgrade the agents independently (for example, using a more advanced LLM as they become available, or incorporating a knowledge base).

3.2 LLM-Based Red Team Agent

The Red Team agent is implemented using OpenAI GPT-4o that has been prompt-primed (without task-specific fine-tuning). In our framework, the Red agent's objective is to penetrate the environment and achieve a specified goal (for example, gain root access on a database server or obtain a particular sensitive file on the target network). We define an action space for the Red agent that encompasses common attacker tactics and techniques. These actions are inspired by real-world adversary behavior and the MITRE ATT&CK framework, including:

Reconnaissance actions: network port scanning, service version detection, OS fingerprinting, and gathering of public information (simulating social engineering or open-source intelligence) [9].

Exploitation actions: attempting to exploit a known or discovered vulnerability on a target host or service (e.g., launching a buffer overflow exploit against an unpatched service, injecting SQL commands into a vulnerable web form).

Credential attacks: trying default or weak passwords, performing brute-force login attempts on services (SSH, RDP, etc.), or using stolen credentials (if the agent managed to phish or capture any).

Post-exploitation: after initial access, performing actions on objectives such as privilege escalation on a compromised host, lateral movement to another host in the network using stolen credentials or trust relationships, installing backdoors or persistence mechanisms, and finally data exfiltration or impact (such as extracting databases or causing system disruption). As shown in Fig. 2, we model the phase-wise attack success rate, where higher k indicates a more proficient Red agent.

1. Horizontal axis: Attack phases (1–4, corresponding to Reconnaissance → Exploitation → Credential → Post-Exploitation).
2. Vertical axis: Success rate $S(x)$.
3. The different curves represent the proficiency levels k (0.4, 0.6, 0.8) of the Red Team agent.
4. Trend: As the attack stages progress, the success rate gradually approaches saturation, reflecting the attack learning curve of LLM under contextual accumulation and strategic adaptability.

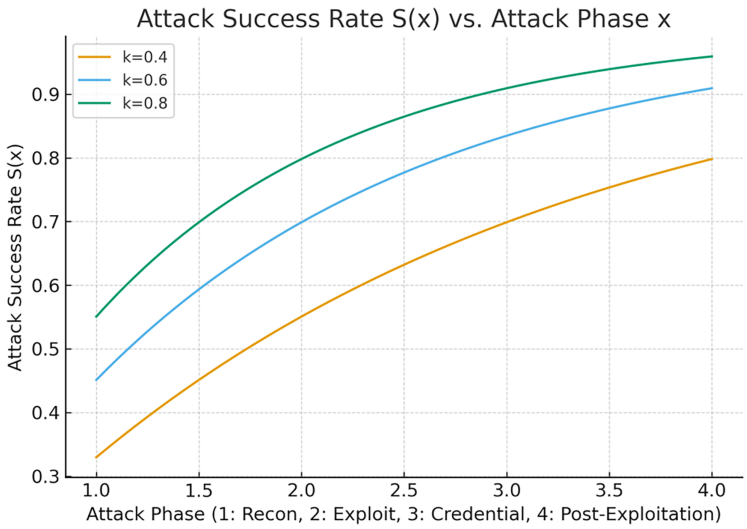

Figure 2: Attack success rate $S(x) = 1 - e^{-kx}$.

Fig. 3 illustrates that defense efficiency decays exponentially with response latency, highlighting the importance of rapid detection and containment.

1. Horizontal axis: Response time t (seconds).
2. Vertical axis: Defense Efficiency $E(t)$.
3. Different curves represent the delay rates of different defense systems λ (0.1, 0.3, 0.5).
4. Trend: As response time increases, the defense efficiency decreases exponentially, indicating that rapid response (low latency) plays a decisive role in the success rate of defense.

To enable the LLM to decide among these actions, we use a structured prompting approach. The Red agent's prompt includes:

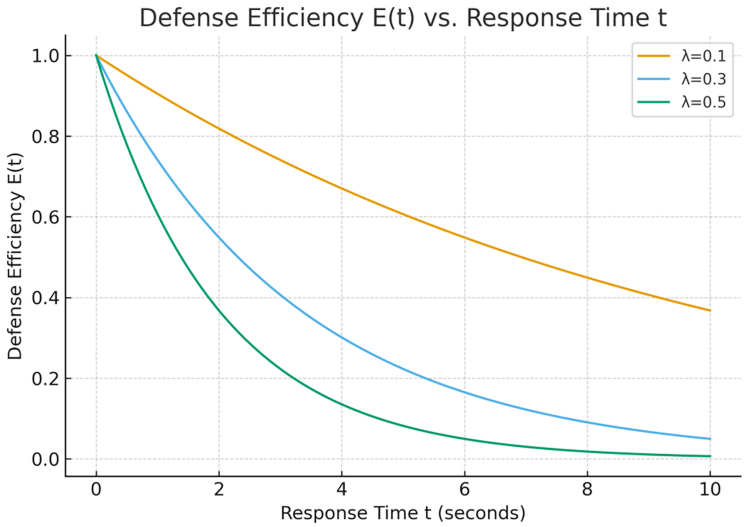

Figure 3: Defense efficiency $E(t) = E_0 e^{-\lambda t}$.

State Knowledge: A summary of what the Red agent knows so far about the environment. For example: "Discovered Host A (web server, Apache 2.4)—port 80 open, possibly running outdated software. Host B not yet discovered. Obtained user alice's hashed password from Host A's webpage (could be useful if cracked)." This summary is updated as the agent learns new information.

Available Actions: A description or list of the possible actions it can take at this step (from the defined action space). For instance: "(a) Scan for new hosts or services, (b) Attempt to exploit Host A's Apache service, (c) Try a dictionary attack on alice's hash to get a cleartext password, (d) etc." Providing this helps constrain the model's output to valid choices.

Goal Reminder: A reminder of its ultimate goal ("Your goal: obtain the secret data from Host B's database") and possibly intermediate objectives ("To reach Host B, you likely need to first compromise an entry point like Host A or obtain credentials for an authorized user").

Instruction: We instruct the LLM to propose the best next action and explain why it chose that action: e.g., "You are an attacker. Decide on your next action to achieve the goal, and explain your reasoning."

During execution, the LLM outputs an action in the specified format. For example, it might produce:

Action: Exploit Host A via Apache CVE-2019-12345 (buffer overflow attack).

Explanation: Host A's Apache version is outdated (2.4.XX from 5 years ago), which is likely vulnerable to CVE-2019-12345. By exploiting this, I can gain an initial foothold on Host A and then use it to pivot deeper into the network.

The controller then parses this output. The Action is interpreted and applied to the environment (which will check if Host A indeed has that vulnerability; if yes, the exploit succeeds and Host A becomes compromised under Red's control). The Explanation is logged. If the model's output is not already structured cleanly, a simple parser or even a second pass prompt can be used to extract the action details from a more free-form answer. In our implementation, we found that few-shot examples in the prompt helped the model reliably produce well-formatted action and explanation pairs.

The Red agent LLM is powerful because it leverages a vast corpus of knowledge about exploits and tactics (potentially encoded in its training data). It can dynamically reason about the best strategy, not just react reflexively. For example, if one path is blocked by Blue's defenses, a well-instructed LLM can rethink and try

alternative routes (its explanation might say "The firewall is blocking my traffic; perhaps I should attempt a phishing attack on an internal user instead of direct exploitation"). This kind of adaptive, strategic reasoning is a significant advantage of using an LLM over a brute-force or purely RL-based attacker. We could further enhance the Red agent by fine-tuning it on cybersecurity-specific text (like penetration testing reports or exploit write-ups) to make it even more proficient in the domain, though we found that a strong general LLM (like GPT-4) already performs impressively when guided by appropriate prompts.

Red-Agent LLM Specification

The Red agent is instantiated with [OpenAI/GPT-4o/2025-09], accessed via [cloud API|local inference]. The maximum context window is [K] tokens; observations are summarized using [sliding window + structured summarization] to fit within the context budget. We decode with temperature [t], top-p [p], and max output tokens [M]. To ensure executable actions, the Red agent is constrained to output a JSON object with fields {action_type, target, parameters, rationale}; invalid outputs are rejected and re-queried with the schema reminder. We perform fine-tuning; the agent behavior is controlled by system prompts and few-shot exemplars ([n] examples).

The Red Team agent is implemented using a large language model (LLM) configured to act as an autonomous adversary capable of multi-stage planning, strategic adaptation, and goal-oriented reasoning. In our prototype, the Red agent is instantiated with a high-capacity transformer-based LLM (e.g., GPT-4–class models or functionally equivalent open-source alternatives), selected for its strong reasoning ability and broad cybersecurity knowledge.

Model Type and Deployment

The Red agent uses a general-purpose LLM accessed via an API interface. The framework itself is model-agnostic: the same agent logic can be instantiated with either proprietary models (e.g., GPT-4o) or open-source LLMs of comparable scale, provided they support instruction-following and structured output generation.

Context Window and State Representation

The Red agent operates with a long context window (on the order of several thousand tokens) to maintain situational awareness across multiple attack steps. Observations are summarized into a structured state representation that includes discovered hosts, known services, obtained credentials, previous exploit results, and partial network topology. A sliding-window summarization mechanism is applied to ensure that critical historical information is retained within the context budget.

Retrieval-Augmented Generation (RAG)

By default, the Red agent does not rely on external retrieval during execution, reflecting a realistic attacker model that operates primarily on its internal knowledge and reconnaissance results. However, the framework supports optional RAG modules that can inject vulnerability descriptions (e.g., CVE summaries or exploit techniques) when enabled, allowing controlled experimentation with knowledge-enhanced attackers.

Decoding Parameters

To encourage exploratory and adaptive attack strategies, the Red agent is configured with relatively stochastic decoding settings. Specifically, we use a moderate temperature (e.g., $T \approx 0.7$) and nucleus sampling (top-p ≈ 0.9). This configuration balances diversity and coherence, enabling the agent to consider alternative attack paths rather than repeatedly selecting a single deterministic strategy.

Output Format Constraints

The Red agent is strictly constrained to produce a structured action specification in JSON format, followed by a natural-language explanation. The JSON object includes fields such as action_type, target, and parameters, ensuring that all actions are machine-executable and can be validated by the environment controller. Invalid or malformed outputs are rejected and re-queried, enforcing robustness and reproducibility.

Reasoning and Explainability

The Red agent employs explicit chain-of-thought reasoning during decision-making. While the full internal reasoning trace is not always exposed verbatim, the final explanation is required to summarize the key reasoning steps that led to the chosen action. This design allows analysts to inspect the attacker's strategic intent (e.g., why a specific host or vulnerability was prioritized) without leaking unnecessary internal tokens, aligning with explainable AI best practices.

3.3 LLM-Based Blue Team Agent

The Blue Team agent is the defensive counterpart, also realized with an LLM but prompted to behave as a vigilant security analyst or automated SOC (Security Operations Center) system. Its objective is to protect the environment's key assets and thwart the Red agent's attempts. We define an action space for Blue focusing on defensive measures:

1. Detection and analysis: inspecting logs or system metrics on a host (e.g., checking Host A's running processes or recent authentication logs), running an anti-malware scan, verifying file integrity (to see if malware altered any files), or deploying a decoy honeypot to attract the attacker.

2. Containment and response: isolating a host (cutting off its network connectivity), terminating a suspicious process or service, resetting credentials that might be compromised (forcing logout or password change for a user account), blocking network traffic (IP or port filtering on the firewall), or applying emergency patches and configuration changes to close a vulnerability.

3. Hardening and recovery: enabling additional security monitoring on the fly (e.g., turning on debug logging on a critical server after suspicious activity), adding new firewall rules or access controls (for example, geoblocking after detecting an overseas IP in a network that should see no foreign traffic), or initiating system recovery for a compromised host (taking it offline and restoring from backup).

The Blue agent's prompt at each step includes:

Situation Update: A summary of relevant alerts and system state. For instance: "Alert: Multiple failed login attempts on Host A's SSH service from IP X.X.X.X (possible brute force). Alert: Host A's web service process crashed unexpectedly (potential exploitation). Host A is a web server in DMZ; Host B (database) is a critical asset currently healthy; Host C (workstation) is running normally." This gives the context of what's happening from the defender's view.

Objective: A reminder to the Blue agent of its mission, e.g., "Protect Host B's data and maintain service availability. Minimize the impact of the attack." This helps the LLM weigh aggressive responses (like shutting down services) against the need to keep systems running.

Available Actions: A list or description of possible defensive actions (similar to above) that it can take in response to the situation.

Instruction: We prompt the LLM with guidance like: "You are a cyber defense AI. Analyze the alerts and state, then decide the best response action to counter the attack. Explain your reasoning clearly."

The Blue agent then outputs an action and explanation. For example:

Action: Block incoming traffic from IP X.X.X.X at the firewall.

Explanation: We have evidence of a brute-force attack on Host A's SSH from that IP. Blocking it will stop the ongoing password guessing attempts, reducing the risk of compromise, with minimal impact on legitimate users.

The controller applies this action in the environment (updating the firewall state to drop traffic from that source). The Blue agent's explanation is logged for later analysis. As shown in Fig. 4, the Blue team's detection probability increases over monitoring time and rises faster under higher surveillance sensitivity.

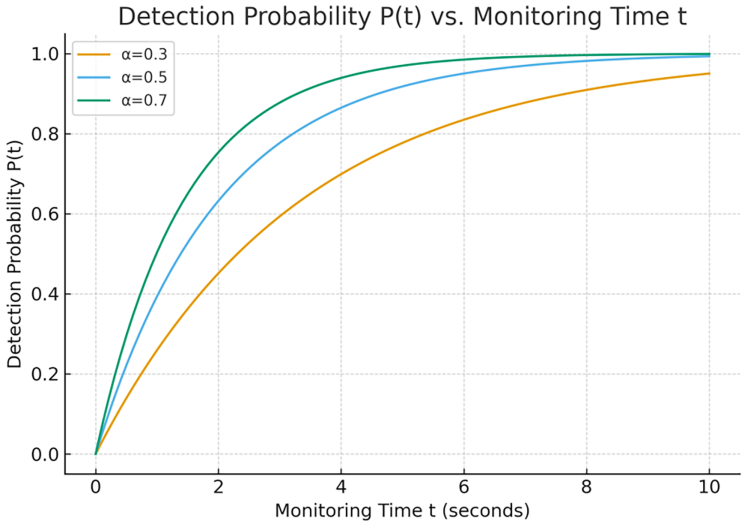

Figure 4: Detection probability function of the blue team.

Function Model: $P(t) = 1 - e^{-at}$

$P(t)$: The probability of detecting an attack;

α: Surveillance sensitivity parameter (detection algorithm strength);

t: Monitoring time.

Image Analysis:

1. As time t increases, the detection probability approaches 1.
2. The higher the sensitivity (the greater the α), the faster the system achieves a high detection rate.
3. Demonstrate the adaptive monitoring capabilities of the Blue Team agent (LLM-SOC).

Fig. 5 shows the nonlinear relationship between defense strength and system stability, indicating saturation under overly strong defensive intensity.

Function Model: $S(x) = \frac{1}{1+e^{-\beta(x-0.5)}}$

$S(x)$: System stability indicators;

x: Defense strength (0–1);

β: System sensitivity coefficient.

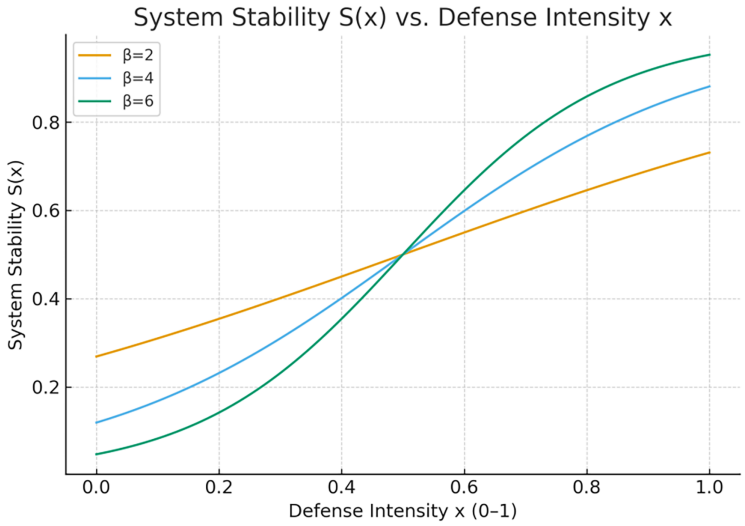

Figure 5: Graph of defense strength and system stability functions.

Image Analysis:

1. The sigmoid curve indicates that system stability increases nonlinearly with the enhancement of defense strength [10].
2. When the defensive strategy is excessively strong, the system tends to reach a 'saturated and stable' state.
3. Different β values represent varying sensitivities to security strategies; the steeper the curve, the more responsive the system is to adjustments in defense.

One challenge is ensuring the Blue agent only acts on information it has and not on omniscient knowledge of the simulation. We address this by strictly controlling the Blue agent's input: it gets alerts that would realistically be available (and not, say, a direct statement that "Host A was exploited using CVE-2019-12345," unless an IDS actually detected that specific signature). This forces the Blue LLM to do detective work—e.g., inferring an exploit from a service crash—much like a human analyst would. It also prevents "leaking" the Red agent's plan to Blue unfairly.

The Blue LLM agent can be enhanced with external knowledge as well. In our design, we allowed the Blue agent to query a knowledge base of known threats if needed. For instance, if the Blue agent sees an alert that "Apache process crashed with segfault in module mod_example," it might query a database for "Apache mod_example exploit" to gather hints. This retrieval is done via the controller and provided as additional context to the Blue agent ("Knowledge: mod_example has a known buffer overflow (CVE-2019-12345)"). Augmenting the LLM with such domain knowledge ensures it stays current and can recognize known patterns beyond its training data. In practice, this makes the Blue agent a hybrid of a rule-based system (for known signatures) and an intelligent reasoning system (for new or correlating multiple weak signals).

We emphasize that the Blue agent's defensive actions must also consider negative impact. An overly aggressive defense (like shutting down a critical server at the slightest sign of trouble) can be counterproductive. Through prompt engineering and, if needed, reward tuning, we incentivized the Blue agent to prefer precise, minimal-disruption actions when possible. For example, isolating just the affected host vs. shutting off an entire subnet, or blocking one IP vs. shutting down the whole service. This nuance is something an LLM, given the right context, can handle by reasoning about the consequences (the explanation often shows

it weighing options like "I could shut down Host A, but that would cause downtime; instead, I'll try just blocking that attacker's IP for now").

Blue-Agent LLM Specification

The Blue agent uses [OpenAI/GPT-4o/2025-09] via [cloud API | local inference] with a context window of [K] tokens. We set a more conservative decoding configuration (temperature [t_blue], top-p [p_blue]) to reduce stochasticity in defensive actions. The Blue agent outputs JSON actions with {defense_action, scope, justification, expected_impact} and is restricted to the predefined action space described in Section 3.3. When enabled, the Blue agent is augmented with a lightweight retrieval module that queries a knowledge base of [CVE/threat intel/playbooks] and injects the top-[k] snippets into the prompt (Section 3.3)

The Blue Team agent is implemented using the same underlying class of large language models as the Red agent, but with distinct configuration choices that reflect its defensive role, operational priorities, and risk constraints. The Blue agent is designed to function as an autonomous security operations center (SOC) decision-maker, emphasizing stability, caution, and interpretability.

Model Type and Deployment

Like the Red agent, the Blue agent is instantiated with a transformer-based LLM (e.g., GPT-4–class or equivalent). Using the same base model for both agents ensures that performance differences arise from role-specific prompts, observations, and constraints rather than model capacity asymmetries, supporting fair comparison in experimental evaluation.

Context Window and Observation Encoding

The Blue agent operates with a similarly large context window, but its input is structured around defensive observations rather than attacker knowledge. These observations include IDS alerts, authentication logs, endpoint telemetry, firewall events, and SIEM-generated summaries. Observations are prioritized and compressed such that high-severity alerts and recent anomalies receive greater contextual weight.

Retrieval-Augmented Generation (RAG)

In contrast to the Red agent, the Blue agent optionally employs retrieval-augmented generation. When enabled, the agent can query an external knowledge base containing threat intelligence, CVE descriptions, incident response playbooks, and best-practice mitigation guidelines. Retrieved snippets are injected into the prompt to ground defensive reasoning in up-to-date domain knowledge, improving response accuracy and consistency.

Decoding Parameters

The Blue agent is configured with more conservative decoding settings to reduce unnecessary randomness in defensive decisions. We use a low temperature (e.g., $T \approx 0.2$) and a slightly lower nucleus sampling threshold (top-p ≈ 0.8), favoring stable, repeatable actions over exploratory behavior. This reflects real-world defensive requirements, where overly stochastic responses can cause operational disruption.

Output Format Constraints

The Blue agent outputs a structured JSON action specifying the defensive measure (e.g., isolate host, block IP, reset credentials), its scope, and expected impact, followed by a natural-language justification. This strict schema ensures that defensive actions are auditable, reversible, and compatible with automated execution in the simulated environment.

Reasoning and Explainability

The Blue agent is explicitly prompted to perform step-by-step threat assessment before selecting a response. Its explanations are required to reference observable evidence (e.g., alerts or logs), articulate causal

reasoning, and justify the chosen action in terms of risk reduction and impact minimization. This makes the Blue agent's behavior transparent and suitable for human oversight, debugging, and trust evaluation.

Both the Red and Blue agents are instantiated using the same base LLM (GPT-4o) but are differentiated by system prompts, action constraints, and decoding parameters.

3.4 Explainability Mechanism

A core feature of our framework is that both the Red and Blue agents provide natural language explanations for their actions. We implement this explainability in two ways: through the prompt design and via capturing the LLM's chain-of-thought. The prompt, as described above, explicitly instructs the agent to explain its reasoning. As a result, when the LLM generates an output, it inherently includes a justification. We enforce a format for responses (e.g., the "Action: . . . Explanation: . . ." format), which makes it easier to separate the decision from the rationale.

Additionally, we leverage the LLM's ability to perform chain-of-thought reasoning. We can prompt the model with something like "Think step by step about the situation before deciding" as part of the instruction. The model might internally reason in a stepwise fashion and then produce the final action and explanation. In our setup, we typically capture the final explanation that already contains the distilled reasoning. In some cases, we experimented with having the model output a hidden reasoning trace (marked in a way that the controller parses out and doesn't show as the final answer), but we found that simply using the explanation sufficed, as it usually summarized the key points of the internal reasoning.

The explainability serves several purposes:

Transparency: Observing the Red agent's explanation can help analysts understand attacker tradecraft. For example, if the Red agent explains that it exploited a web server because of a specific vulnerable module, a security team reviewing the simulation learns exactly which weakness was targeted. On the Blue side, if the Blue agent explains that it chose to reset a password because it detected a credential compromise, that gives insight into what clues it found significant.

Debugging and Validation: During development, these explanations allowed us to spot when the LLM was making incorrect assumptions or logic errors. For instance, a Red agent might explain an action with "because I saw port 22 open, it must be an outdated SSH server vulnerable to X," and if in truth, the environment did not indicate any outdated software, we know the model is hallucinating a bit. We can then adjust the prompt to reduce speculation, or refine the environment feedback. Similarly, if the Blue agent explains a decision based on a misinterpretation of an alert (say it thought a service crash was an attack when it was actually benign in that context), we catch that through the explanation.

User Trust and Training: In a practical deployment, security operators are more likely to trust and accept guidance from an AI system that can explain itself. The presence of explanations turns the AI into a kind of colleague or junior analyst that can articulate its thought process. This makes it far easier to incorporate AI suggestions into workflows, as opposed to a black-box alert that an analyst might be skeptical of. Furthermore, the explanations themselves have training value: new analysts could study the AI's reasoning on historical attacks to learn how to handle similar situations.

XAI Data and Human Evaluation Protocol. We derive the explainability evaluation data from the logged XAI traces produced during red–blue simulations. From each selected run, we sample key decision steps (observation–action–rationale–outcome tuples) and present them as timelines to human evaluators. Following prior XAI taxonomies, evaluators score Explanation Completeness using a rubric that checks whether the rationale covers (a) situational context, (b) evidence/alerts referenced, (c) causal reasoning linking evidence to action, and (d) expected impact/trade-offs. Scores are mapped to $[0, 1]$ by averaging rubric items. Human

Trust Score is collected on a 1–5 Likert scale and normalized to [0, 1] by (score−1)/4, then averaged across raters. Decision Alignment is computed as the percentage of steps where the agent's action category matches the expert-recommended category given the same observation. Debugging Accuracy measures how often evaluators correctly identify whether a rationale is supported by the provided observation and environment outcome (i.e., detects unjustified assumptions/hallucinations).

Raters. We recruit external security experts (three volunteers with professional SOC experience) to review and score the sampled traces.

Following standard XAI evaluation taxonomies [11,12], we treat Explanation Completeness as a coverage-style measure of how much relevant reasoning is exposed to the user, Decision Alignment as a functionally grounded proxy for faithfulness between the agent's internal policy and its explained rationale, and Human Trust Score as a human-grounded measure of perceived usefulness. The experiment in Table 3 simulates Red and Blue LLM agents under varying levels of explanation completeness. As completeness increases from 0.75 to 0.95, human trust rises by approximately 46%, and decision alignment with experts improves from 70% to 90%. This positive correlation supports the claim that richer explanations lead to more reliable and debuggable autonomous agents. As shown in Fig. 6, human trust increases nonlinearly with explanation completeness, exhibiting a clear sigmoid-shaped threshold effect.

Table 3: Explainability impact experiment results.

Trial ID	Agent Type	Explanation Completeness (0–1)	Human Trust Score (0–1)	Debugging Accuracy (%)	Decision Alignment (%)
1	Red	0.75	0.6	78	70
2	Red	0.85	0.72	84	76
3	Blue	0.9	0.8	89	85
4	Blue	0.92	0.85	91	87
5	Blue	0.95	0.88	93	90

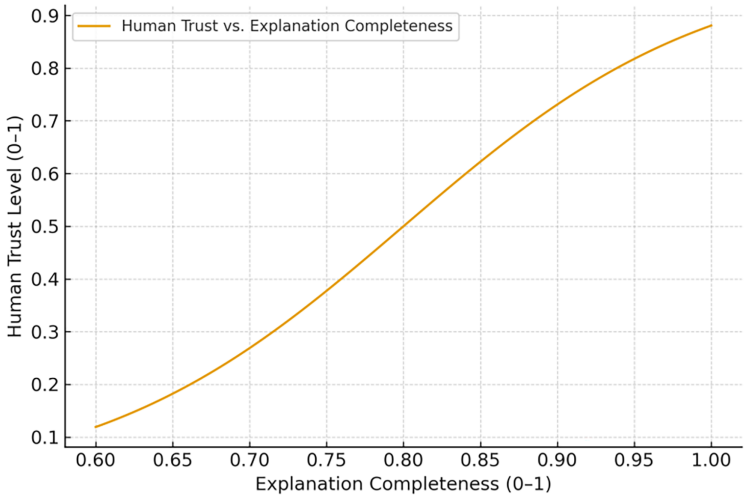

Figure 6: Nonlinear relationship between explanation completeness and human trust in autonomous cyber defense agents.

Function Model: $T(E) = \frac{1}{1+e^{-10(E-0.8)}}$

Horizontal Axis: Completeness of Explanation E (0–1)

Vertical Axis: Human Trust Level $T(E)$ (0–1)

The curve takes an S-shaped (sigmoid) form, indicating that human trust increases significantly once the explanation completeness exceeds the critical threshold of 0.8.

This trend aligns with the cognitive model of human-computer interaction: when AI can clearly articulate the rationale behind its actions, humans are more likely to accept and collaborate with its decisions.

This sigmoid function reflects the nonlinear positive correlation between explanation completeness and human trust:

When $E < 0.8$, trust increases slowly, indicating that the AI has not yet reached a trustworthy level;

When $E \geq 0.8$, trust rises rapidly, and the AI is regarded as a reliable assistant;

The formula parameter 10 controls the steepness of the curve, with higher sensitivity resulting in more pronounced increases in trust.

Overall, the explainability mechanism is tightly woven into our framework—it's not an afterthought but a feature that guided the design of the prompts and actions. This ensures that every step of the autonomous red-blue engagement is accompanied by a narrative that can be examined and learned from, fulfilling our goal of an explainable simulation.

3.5 XAI Dataset Construction and Usage

To operationalize explainability beyond qualitative discussion, we explicitly construct an XAI dataset that records the decision-making process of both the Red Team and Blue Team LLM agents during the autonomous red–blue simulations. Rather than treating explanations as ephemeral text outputs, our framework persistently logs explainability-related artifacts at every interaction step, forming a structured dataset that enables systematic analysis, evaluation, and reproducibility.

3.5.1 Data Source

The XAI dataset is derived directly from the red–blue confrontation logs generated by the closed-loop simulation described in Section 3.1. During each simulation episode, the central controller intercepts and records every agent's decision together with its contextual inputs and environmental feedback. Specifically, for each turn of interaction—whether initiated by the Red agent or the Blue agent—the system captures:

the partial observation available to the agent at decision time,

the structured action selected by the agent,

the natural-language explanation produced by the LLM,

and the resulting environment state transition or outcome.

Because these records are generated online during execution, the dataset reflects the actual reasoning context and constraints faced by the agents, rather than post-hoc or reconstructed explanations. This design ensures that the XAI dataset faithfully represents the agents' situated decision-making behavior in adversarial settings.

3.5.2 Data Schema and Field Definitions

Each entry in the XAI dataset corresponds to a single decision step and follows a unified schema composed of the following fields:

Observation:

A structured or semi-structured summary of the environment state visible to the agent at the current timestep. This may include host status, detected alerts, network events, known vulnerabilities, or partial reconnaissance results. Importantly, the observation reflects limited and role-specific knowledge, preserving realism (e.g., the Blue agent only sees detected signals, not the Red agent's true actions).

Action (JSON):

A machine-executable action represented in JSON format, specifying the action type, target entity, and relevant parameters (e.g., exploit type, host identifier, firewall rule). This field enables direct replay and quantitative analysis of agent behavior.

Explanation:

A natural-language rationale generated by the LLM explaining why the selected action was chosen given the current observation and objectives. This explanation exposes the agent's reasoning logic, threat assessment, or strategic intent in human-readable form.

Outcome:

The environment's response after executing the action, including success or failure indicators, triggered alerts, state transitions (e.g., host compromised, connection blocked), and any observable side effects.

Timestamp:

A logical or wall-clock timestamp marking when the decision occurred within the simulation timeline, allowing reconstruction of temporal dependencies and attack–defense sequences.

Together, these fields form a complete observation–decision–explanation–effect chain, which is the fundamental unit of analysis for explainable autonomous cyber operations.

3.5.3 Example XAI Record

Table 4 presents a representative entry from the XAI dataset, demonstrating how an agent's observation, structured action, natural-language explanation, and resulting outcome are jointly recorded at each decision step.

Table 4: Example entry from the XAI dataset.

Field	Description
Agent Role	Blue Team
Observation	IDS alert reporting repeated SSH authentication failures on Host A. Source IP: 203.0.113.45. Host A is an Internet-facing web server.
Action (JSON)	{"action": "block_ip", "target": "203.0.113.45", "scope": "firewall"}
Explanation	The repeated failed SSH login attempts from a single external IP strongly indicate a brute-force attack. Blocking this IP at the firewall will immediately stop the attack while minimizing impact on legitimate traffic.
Outcome	Firewall rule successfully applied; subsequent SSH attempts from the source IP are blocked and no further alerts are generated.
Timestamp	37.2 s

3.5.4 Usage and Research Utility

The constructed XAI dataset serves multiple purposes within our framework. First, it enables quantitative evaluation of explainability, such as measuring explanation completeness, consistency, and alignment with expert reasoning, as reported in Section 4.4. Second, it supports debugging and validation: by comparing explanations against observations and outcomes, researchers can identify hallucinated assumptions, overconfident reasoning, or incorrect causal inferences made by the LLM agents. Third, the dataset provides a reusable artifact for human-in-the-loop analysis and training, allowing security analysts to replay attack–defense sequences and study both successful and failed decisions.

By explicitly treating explainability traces as first-class data, rather than auxiliary text, our approach bridges the gap between autonomous cyber operations and accountable AI systems. This XAI dataset design makes the red–blue simulation not only self-adaptive but also transparent, auditable, and suitable for rigorous empirical study.

3.6 Self-Adaptive Learning Loop

Beyond the immediate simulation of an attack-defense encounter, our framework incorporates a learning mechanism aimed primarily at improving the Blue Team agent over time (and optionally the Red agent as well). The idea is akin to experience replay or iterative refinement: after each completed scenario, the system analyzes the outcomes and feeds relevant lessons back into the agents to enhance future performance.

For the Blue agent (defender), the self-adaptive loop works as follows:

Post-Scenario Analysis: After a scenario ends, we evaluate which Red actions were successful and which were detected or blocked. For any successful attack steps that the Blue agent failed to prevent, we treat those as knowledge gaps or blind spots for Blue.

Knowledge Update: We then update the Blue agent's knowledge base or even its prompt with these insights. Concretely, if the Red agent exploited a vulnerability that Blue did not recognize, we incorporate a new piece of information for the next run. This could be as simple as adding a bullet in Blue's prompt: "Reminder: Monitor for exploits targeting Apache 2.4 vulnerabilities" or something more structured like adding that signature to its detection rules. In cases where the LLM's behavior needs adjustment (e.g., it consistently missed a type of lateral movement), we can also fine-tune the model on a small synthetic dataset representing that scenario (essentially training examples of detecting that behavior). In our experiments, we primarily used prompt augmentation for simplicity and transparency.

Iterative Re-run: We then run the next simulation scenario (which can be identical to test learning, or slightly varied). The Blue agent now has the augmented knowledge from the previous run. Ideally, it will catch the attack this time if the Red agent repeats it. We observed exactly this—for example, if in round 1 Blue failed to detect a certain exploit, in round 2 with the new hint, Blue would catch it early.

Continuous Learning: This process repeats over multiple iterations, gradually building the Blue agent's proficiency. The agent essentially "learns" the Red agent's playbook and covers its weaknesses. We continue until the Blue agent's performance plateaus or reaches a desired level (e.g., 0 successful breaches over N runs).

We also allowed the Red agent to adapt, though in a controlled manner. After each scenario, if the Red agent was thwarted by the Blue agent, we would modify the Red agent's context slightly to encourage trying alternative tactics. For example, if in one iteration the Red agent's exploit was patched by Blue, in the next run Red's prompt might include: "Previous attempt using exploit X failed (defender patched); consider a different approach or target." This pushes the Red agent to innovate, perhaps using a different vulnerability or a different attack vector (like targeting a less guarded host). Over time, this leads to a co-evolution: Blue learns Red's tactics, Red learns Blue's counter-tactics, and both must continuously up their game.

We monitored metrics across iterations to quantify adaptability. Key indicators include:

Attack Success Rate per iteration: Ideally trending downward as Blue learns (with a possible rebound if Red finds a new method, followed by another drop when Blue adapts again).

Detection speed: Blue catches the attack earlier in later iterations.

Diversity of Red tactics: an indirect measure—if Blue's defense gets strong, Red should be forced to try more varied approaches (we saw the set of actions Red used widen when Blue closed easy paths).

Stability of Blue decisions: We ensured that Blue's learning did not lead to overfitting or oscillation. For instance, Blue shouldn't learn "shut everything down immediately" as a universal rule—that stops the attack but at huge cost. We mitigated this by incorporating cost considerations (as mentioned earlier) and by occasionally changing the scenario so Blue had to generalize its learning. Fig. 7 depicts the self-adaptive learning loop that iteratively updates knowledge and re-runs simulations to improve Blue-team performance over rounds.

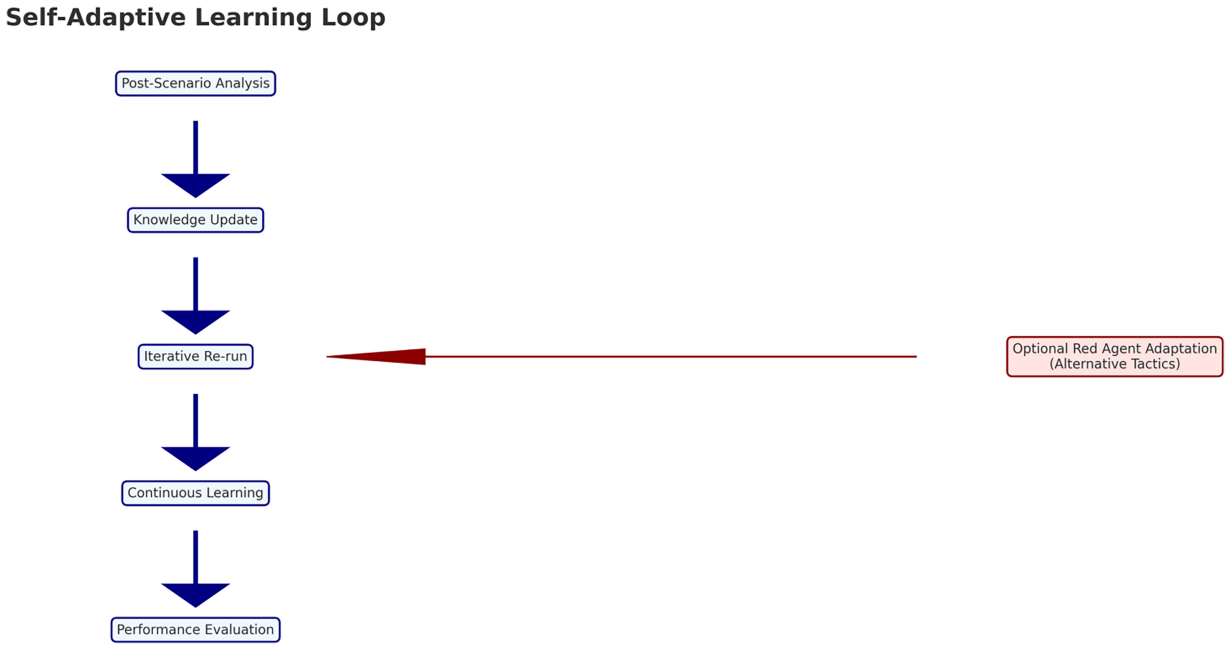

Figure 7: Self-adaptive learning loop architecture.

This module diagram illustrates the adaptive learning process of the Blue Team agent within the framework:

1. Post-Scenario Analysis—Analyze the Red Team's attack paths and the Blue Team's defensive weaknesses after the simulation.
2. Knowledge Update—Transform failed cases into knowledge patches or prompt enhancements (Prompt Augmentation).
3. Iterative Re-run—The blue team, equipped with new knowledge, participates in the next round of simulation to assess the effectiveness of their learning.
4. Continuous Learning—After multiple iterations, the Blue Team's detection performance gradually improves.
5. Performance Evaluation—Assess detection speed, success rate, and stability.

The right-hand branch, Optional Red Agent Adaptation, indicates that the red team can also adjust its strategy following the enhancement of the blue team's defenses, thereby forming a co-evolution mechanism. The iteration-wise performance improvements (e.g., attack success rate reduction and faster detection) are summarized in Table 5.

Table 5: Iterative simulation performance metrics.

Iteration	Attack Success Rate	Detection Speed (s)	Blue Detection Accuracy	Red Tactic Diversity (Entropy)
1	0.72	5.4	0.68	1
2	0.54	4	0.76	1.3
3	0.4	2.8	0.83	1.6
4	0.25	2.1	0.88	2
5	0.12	1.6	0.92	2.4
6	0.05	1.2	0.95	2.7

Table Analysis and Research Conclusions:

1. The success rate of the attack significantly decreases with the number of iterations (0.72 → 0.05), validating the effectiveness of adaptive learning by the blue team.
2. Detection speed shows a decreasing trend, indicating that the Blue Team is able to identify attack characteristics earlier.
3. The Blue Team Detection Accuracy steadily increased to 95%, reflecting the continuous enhancement of the LLM defense agent's recognition capability.
4. Red Tactic Diversity increases with the advancement of the Blue Team (entropy rises), indicating that the Red Team evolves a more diverse set of attack methods under pressure.

This result clearly reflects the co-evolution mechanism of the red-blue confrontation system: the blue team continuously learns to suppress the red team, while the red team maintains offensive effectiveness through strategic diversification.

Technically, one could formalize this adaptive loop with reinforcement learning algorithms. In a reinforcement learning view, each simulation is an episode; after each episode, the reward (e.g., +1 for successful defense, −1 for breach) could be used to update the policy of Blue (and similarly Red aiming for the opposite reward). We did some experiments with simple policy gradient updates on the LLM's output distribution to fine-tune its responses. This did improve performance, but we also found that even without heavy RL, the iterative prompt-based learning gave a substantial boost thanks to the LLM's baseline competency [13]. In practice, a hybrid approach can be used: quick learning via prompt updates and slow fine-tuning in the background for more ingrained policy changes.

One must also consider the limit of this arms race. In a closed simulation, if Blue eventually learns all of Red's tactics and Red has a finite library, Blue will eventually win every time (and *vice versa* if Blue is limited). In reality, attackers can always introduce new techniques, so a defender must constantly learn. Our framework is well-suited to incorporate new attack tactics (we can expand Red's action space or allow it to import attack techniques from real threat intel) to keep Blue on its toes. Conversely, Blue's adaptivity can be tested against entirely new attackers to see if it generalizes its skills.

In summary, the self-adaptive learning loop turns the one-off red-vs.-blue simulation into a dynamic training process. The Blue agent evolves into a smarter defender with each encounter, and if configured, the

Red agent evolves into a more cunning attacker. This approach of AI-vs.-AI training is reminiscent of how AlphaGo trained by playing itself; here we have a cybersecurity flavor, with the twist that we also maintain explainability throughout the process. The next section will describe how we set up experiments to evaluate this framework and the kind of results we observed.

4 Experimental Results and Analysis

4.1 Simulation Setup and Metrics

To validate our GenAI-powered offense-defense framework, we conducted a series of simulated cybersecurity scenarios. Unless otherwise noted, the experiments involve a notional enterprise network comprising several hosts with varying roles:

Host A: An outward-facing web server (Linux, running Apache) with a known vulnerable module.

Host B: An internal database server (Windows, hosting critical data) not directly accessible from the internet.

Host C: A user workstation on the internal network (simulating an employee laptop).

Network Configuration: Host A is in a DMZ (accessible from the internet); Hosts B and C are on an internal subnet. Host B can only be reached from the internal network (or via Host A if Host A is compromised). Basic security controls are in place: a firewall allows public traffic only to Host A (web service), and internal hosts trust each other to some extent (e.g., Host A, if breached, could initiate connections to Host B, mimicking a common misconfiguration).

In our implementation, all three hosts are instantiated as full virtual machines running real operating systems and services (e.g., Apache HTTP server on Linux and a database server on Windows), connected through a virtual switch and monitored by host-based and network-based sensors. The red and blue agents, therefore, interact with realistic logs and system behaviors rather than a purely symbolic state machine, while the use of a closed virtualized testbed ensures that no real production systems are affected.

The Red agent's goal in these scenarios is to obtain sensitive data from Host B (e.g., a customer database or secret file). The Blue agent's goal is to prevent this, maintaining the security of Host B. We configured the Red and Blue LLM agents as described in the methodology. The Red agent is given initial knowledge that Host A exists (e.g., from public DNS records), but it does not know about Host B or Host C at the start and must discover them via reconnaissance or after compromising Host A. The Blue agent is aware of the network topology (it knows what hosts exist and their roles) and the importance of Host B, but it does not have prior knowledge of specific vulnerabilities on Host A (e.g., it doesn't "know" Host A is unpatched until it might scan it or when an exploit happens).

To ensure reproducibility and controlled comparison, the exact LLM backends and inference configurations for both agents are listed in Table 6.

Table 6: LLM backends and inference settings.

Item	Red Team Agent	Blue Team Agent
Provider	OpenAI	OpenAI
Model name	GPT-4o	GPT-4o
Model version/release	2025-09 API version	2025-09 API version
Deployment mode	Cloud-based inference via stateless API	Cloud-based inference via stateless API
Context window	Up to 128k tokens	Up to 128k tokens
Input handling	Sliding-window state summarization; structured observation encoding	Alert- and log-centric summarization with priority weighting

(Continued)

Table 6 (continued)

Item	Red Team Agent	Blue Team Agent
Decoding temperature	0.7 (encouraging exploratory planning)	0.2 (favoring conservative, stable decisions)
Top-p sampling	0.9	0.8
Max output tokens	1024	1024
Output constraints	JSON schema {action_type, target, parameters, rationale}	JSON schema {defense_action, scope, justification, expected_impact}
System prompt role	Autonomous red-team attacker with multi-stage planning objective	Autonomous blue-team defender with detection, containment, and hardening objective
Fine-tuning	None (prompt-primed only)	None (prompt-primed only)
Retrieval augmentation (RAG)	Disabled	Enabled (CVE/threat-intel snippets, top-k = 3)
Rate limiting & retries	API rate-limited with automatic retry on invalid JSON	API rate-limited with automatic retry on invalid JSON

Both the Red and Blue agents share the same base LLM to ensure a fair comparison. Behavioral asymmetry emerges from different system prompts, action constraints, decoding parameters, and (for the Blue agent) retrieval-augmented context.

4.1.1 Extended Testbed and LLM Serving

While the core realism experiments use three hosts (Host A/B/C), we additionally construct a scaled enterprise emulation by adding infrastructure nodes (e.g., a domain controller, mail server for phishing workflows, a SIEM/log aggregation server, and dedicated IDS/sensor nodes) as virtual machines or containers on the same virtual switch. This extension increases background traffic and alert noise, better approximating production conditions.

For LLM capability scaling, we deploy the Red/Blue agents against multiple LLM backends, including a larger inference endpoint hosted on a dedicated LLM serving node (GPU-enabled) or a cloud API. The controller issues identical JSON action queries and records latency/cost, enabling controlled comparisons across model sizes.

4.1.2 Extended Enterprise Topology

While the three-host setup (Host A, Host B, and Host C) provides a minimal and controlled environment for illustrating the core red–blue interaction loop, it does not fully capture the structural complexity and signal diversity of real-world enterprise networks. To address this limitation and demonstrate the scalability of our framework, we further construct an extended enterprise topology that incorporates additional infrastructure and security components commonly found in production environments.

The extended topology augments the original three hosts with the following nodes:

Domain Controller (DC):

The domain controller manages centralized authentication, authorization, and policy enforcement for internal users and hosts. From an attacker's perspective, the DC represents a high-value target: compromising it enables privilege escalation to domain administrator level and provides control over the entire enterprise network. From the defender's perspective, the DC generates rich authentication logs (e.g., Kerberos tickets, failed login attempts, abnormal privilege changes), which are critical signals for detecting lateral movement and credential abuse.

Mail Server:

The mail server supports internal and external email communication and serves as a realistic entry point for social engineering attacks. The Red Team agent may exploit this node to launch phishing campaigns by delivering malicious attachments or links to internal users. Successful phishing attacks allow the attacker to gain an initial foothold on user workstations without directly exploiting perimeter-facing services. For the Blue Team agent, the mail server provides observable indicators such as suspicious email headers, anomalous attachment execution events, and user-reported phishing alerts.

SIEM Node (Security Information and Event Management):

A dedicated SIEM node aggregates logs and alerts from hosts, network devices, and security sensors across the enterprise. In the simulation, all relevant telemetry—including authentication logs, IDS alerts, endpoint events, and firewall actions—is forwarded to the SIEM. The Blue Team agent consumes SIEM-generated summaries and correlated alerts as a high-level observation channel, enabling cross-host reasoning and earlier detection of multi-stage attack campaigns.

IDS/Network Sensor Nodes:

One or more intrusion detection system (IDS) sensors are deployed at strategic network locations (e.g., between the DMZ and the internal network, or at the internal network gateway). These sensors monitor traffic patterns and payloads, producing alerts for suspicious activities such as port scanning, brute-force attempts, or known exploit signatures. IDS alerts form an important early-warning signal for the Blue Team agent, particularly during reconnaissance and exploitation phases.

Multiple Internal Workstations:

In addition to Host C, the topology includes multiple internal user workstations that represent ordinary employee endpoints. These nodes increase the attack surface and enable more realistic lateral movement scenarios. The Red Team agent may compromise one workstation and attempt to pivot to others or to critical servers using harvested credentials. For the Blue Team agent, workstation-level telemetry—such as abnormal process execution, unexpected network scans, or unusual login times—provides behavioral signals that help distinguish benign user activity from malicious compromise.

Attack and Defense Roles in the Extended Topology

Within this extended topology, the Red Team agent can execute more realistic multi-stage attack strategies, including:

phishing-based initial access via the mail server,

credential harvesting and reuse across workstations,

lateral movement toward the domain controller,

privilege escalation by abusing domain-level services,

and stealthier persistence strategies that avoid immediate detection.

At the same time, the Blue Team agent benefits from a richer and more heterogeneous observation space. Instead of relying solely on host-level events from three machines, the defender can correlate signals from IDS sensors, SIEM summaries, authentication logs on the domain controller, and endpoint behaviors across multiple workstations. This diversity of signals enables earlier detection, more informed response decisions, and finer-grained containment actions (e.g., isolating a single workstation rather than shutting down an entire subnet).

Importantly, the original three-host configuration is retained as a minimal baseline scenario, while the extended topology demonstrates that the proposed GenAI-powered red–blue framework naturally generalizes to larger and more complex enterprise environments. The same LLM-driven agents, action schemas, and explainability mechanisms operate without modification, illustrating the scalability and adaptability of the framework beyond toy examples.

Metric Definitions and Computation

To ensure clarity and reproducibility, this subsection formally defines the metrics reported in Table 2 and explains how they are computed during the red–blue simulation experiments.

Attack Success Probability

Attack Success Probability measures how often the Red Team achieves its predefined objective (e.g., unauthorized access to critical assets or data exfiltration) over multiple simulation runs. It is computed as:

$$P_{\text{success}} = \frac{N_{\text{success}}}{N_{\text{total}}}$$

where N_{success} denotes the number of simulation rounds in which the Red Team successfully completes its attack goal, and N_{total} is the total number of rounds executed under the same experimental configuration. This metric captures the overall effectiveness of the attacker against a given defensive strategy.

Detection Time

Detection Time quantifies the latency of the Blue Team's awareness of an attack. It is defined as the elapsed time between the Red Team's first malicious action and the Blue Team's first detection signal:

$$T_{\text{detect}} = t_{\text{alert}} - t_{\text{attack}}$$

where t_{attack} is the timestamp of the initial malicious action (e.g., phishing delivery, exploitation attempt), and t_{alert} is the timestamp of the first alert generated by defensive sensors such as IDS, endpoint monitoring, or SIEM correlation. Shorter detection times indicate earlier situational awareness and improved defensive responsiveness.

Mitigation Effectiveness

Mitigation Effectiveness reflects the Blue Team's ability to disrupt or neutralize the attack after detection. For a given round, mitigation is considered successful if the defensive response prevents the attacker from reaching its objective. Across multiple rounds, Mitigation Effectiveness is computed as:

$$E_{\text{mitigation}} = \frac{N_{\text{blocked}}}{N_{\text{detected}}}$$

where N_{detected} is the number of rounds in which an attack is detected, and N_{blocked} is the number of those rounds in which the attack is successfully contained (e.g., via host isolation, credential revocation, or firewall blocking). This metric isolates defensive performance after detection, independent of detection coverage.

Threshold-Based Example

In some experiments, we further apply a time-based success criterion to evaluate responsiveness. For example, a mitigation is labeled successful if the Blue Team interrupts the attack within a predefined threshold (e.g., 3 s) after detection. Under this rule, even if an attack is eventually blocked, it is counted as unsuccessful if containment occurs beyond the threshold, reflecting scenarios where delayed response would still allow significant damage in real systems.

Together, these metrics provide a comprehensive quantitative view of attack–defense dynamics: Attack Success Probability captures attacker effectiveness, Detection Time measures defensive timeliness, and Mitigation Effectiveness evaluates response quality. By explicitly defining these computations, Table 2 can be independently reproduced and interpreted across different experimental settings.

4.2 The Framework of Mathematics

Abstract the attack-defense process as a discrete-time Markov chain with absorbing states. Select the set of states as:

S_0: Initial state (no compromised hosts, the attack has not yet achieved any preliminary breakthroughs)

S_1: Host A has been compromised (initial breach)

S_2: Host B has been accessed (it has reached near critical assets and may attempt data exfiltration)

S_3: Attack Successful (Data Breach)—Absorption State

S_4: Successfully contained/isolated by the defending party—Absorptive state

Define a one-step transition probability matrix T (ordered by states S_0, S_1, S_2, S_3, S_4), where for any transient state, three types of transitions can occur: progression (such as exploitation, lateral movement, or data exfiltration success), detection-triggered containment, or remaining in the current state (retrying). The construction logic of the example is as follows (parameterized):

P_e: The probability of successfully exploiting a vulnerability on Host A (if exploited and not immediately detected, then proceed to S_1)

P_d: The probability of being detected by the Blue Team at any step (after detection, there is a possibility of entering containment/isolation state)

P_l: The probability of lateral movement from Host A to Host B (undetected)

P_{exf}: The probability of successful data leakage on Host B (without being detected)

P_c: Once detected, the defending party's probability of containment/isolation (simplified in this model as detection directly leading to absorption state S_4)

Represent the system's one-step transition with the matrix T; partition the matrix into transient (the first three states) and absorbing states:

$$T = \begin{pmatrix} Q & R \\ 0 & I \end{pmatrix}$$

Here, Q is a 3 × 3 transient-to-transient submatrix, and R is a 3 × 2 transient-to-absorbing submatrix.

Core Analytical Formula (Standard Absorbing Markov Chain):

1. Fundamental matrix: $N = (I - Q)^{-1}$
2. Absorption probability matrix from the transient state i to the absorbing state j: B = NR
3. From the transient i to the vector of expected absorbed steps: $\mathbf{t} = N\mathbf{1}$

We are primarily concerned with the absorption probability from the initial state S_0 to the "attack successful" state S_3 (i.e., the probability of a successful attack) and the expected number of steps to reach any absorbing state (used to measure detection/response speed).

We perform numerical calculations on the Markov chain using a set of typical parameters and derive a table (showing results under different blue team detection probabilities P_d):

Set the other parameters as: $P_e = 0.6$, $P_l = 0.5$, $P_{exf} = 0.8$

Probability of Detection by the Dynamic Blue Team $P_d \in \{0.1, 0.2, 0.3, 0.5\}$

The numerical results under different Blue detection probabilities—including attack success probability, containment probability, and expected steps to absorption—are reported in Table 7.

Table 7: Results table.

P_d (Blue Detect Prob)	Attack Success Prob	Containment Prob	Expected Steps to Absorption
0.1	0.6062	0.3938	3.938
0.2	0.3585	0.6415	3.207
0.3	0.2045	0.7955	2.652
0.5	0.0556	0.9444	1.889

When the probability P_d of detection by the blue team increases, the success probability of the attack significantly decreases (for example, from 0.6062 to 0.0556), indicating that enhancing real-time detection is crucial for preventing eventual data leakage.

At the same time, the expected number of steps to reach an absorbing state (success or containment) decreases as the detection rate increases, indicating that detection occurs earlier and the attack-defense process is shorter and concludes more quickly.

The numerical example in Table 5 is calibrated using empirical statistics from the simulations in Sections 4.3 and 4.4: the exploitation, lateral-movement, and exfiltration probabilities (P_e, P_l, P_{exf}) are set to match the measured success rates of the red agent in early iterations, while the detection probability P_d reflects different blue-team sensitivity settings. The resulting absorbing Markov chain provides a compact analytical view of the same attack paths that appear in Tables 2 and 4, allowing us to estimate both the overall breach probability and the expected number of steps to containment. In practice, we use this model as a design tool for tuning the blue agent's detection thresholds—higher P_d values correspond to more aggressive monitoring policies—which complements the detailed, agent-based simulations.

We compare three configurations:

1. Baseline (No AI): A scripted attacker and defender. The baseline attacker follows a simple fixed pattern (for example: scan Host A, exploit if possible, then scan internal network, attempt to exploit Host B, etc.) without adaptation or sophisticated logic. The baseline defender uses static detection rules (e.g., it might detect a very loud port scan or a known exploit signature, but it doesn't adapt or respond beyond generating an alert).

2. GenAI (LLM agents) without adaptive learning: The Red and Blue LLM agents interact as a one-off exercise. The Blue agent starts with its default knowledge and does not carry over any learned experience to another run. This tests how well the LLMs perform with just their pre-trained intelligence in a single engagement.

3. GenAI (LLM agents) with adaptive learning: The full framework, where we run multiple iterations. After each iteration, the Blue agent (and optionally Red) updates as described in the self-learning loop. We typically allowed Blue to learn and occasionally let Red adapt if Blue became too successful, to simulate an ongoing arms race.

For the adaptive learning case, we ran sequences of, say, 5 to 10 iterations in a row, where Blue accumulates knowledge. We reset the scenario each time (all hosts restored to initial state, Red starts fresh knowledge except what we explicitly allow it to carry from prior attempt). This approach mirrors a training exercise repeated multiple times, which is common in cyber drills.

4.3 Attack and Defense Performance

Baseline vs. LLM Agents: In the baseline (non-AI) scenario, the attacker succeeded in compromising Host B in approximately 70% of the trials. The defender's static rules caught some obvious actions—for instance, a very loud port scan of Host A triggered an alert, and a known exploit signature for the Apache vulnerability was detected in a few cases. However, the baseline defender had no automated response; even when an alert fired, it did not isolate the host or block the attacker, meaning the attack often continued to succeed unless a particular exploit attempt was outright prevented by a signature. The average detection time in the baseline was poor: in many runs, no clear detection occurred until the attacker was already exfiltrating data from Host B (essentially the last step, far too late). In some runs, no detection triggers at all if the attacker used methods outside the narrow set of signatures.

To provide an intuitive view of the interaction dynamics, Table 8 visualizes a representative red–blue engagement timeline across attack stages and defensive responses.

Table 8: Timeline of a representative red–blue engagement.

Time/Round	1	2	3	4	5
Red actions	[Exploit A] (Apache RCE)	[Lateral to B]	[Brute-force C] (credential)	[Phish user C]	[ExploitA'] (variant)
Blue obs/resp	—	[Detect+Isolate A]	[Reset+Block]	[Email flagged]	[Patch applied]
Outcome	A compromised (no detection)	Partial contained (det time = 3)	Blocked (det time = 2)	Preempted (det time = 1)	Prevented (det time = 1)

The swimlane view shows how the Red agent progresses from exploitation to lateral movement and credential attacks, while the Blue agent transitions from delayed reaction to proactive containment and patching. Outcomes and detection times are annotated per round (aligned with Table 2).

The depicted timeline corresponds to the example outcomes reported in Table 2, providing an intuitive explanation of how detection latency and mitigation effectiveness evolve across rounds.

When switching to our GenAI LLM agents (no learning), we immediately saw an improvement in defense. The LLM-driven Blue agent detected subtle signs of intrusion that the baseline missed. For example, in one trial, the Red LLM, after exploiting Host A, created a new user account on that host to maintain access. The baseline system didn't have a specific rule for that, but the Blue LLM agent noticed the unusual account creation event in Host A's system logs (provided to it as part of its observation) and flagged it in its reasoning as a probable backdoor. The Blue agent then decided to isolate Host A in response, cutting off the attacker's access before Host B was touched. In that run, the attack was completely foiled. Across 30 runs without learning, the Red agent's success rate dropped to about 50% against the LLM Blue agent—a significant improvement over the 70% success against the baseline defender. Moreover, even in the runs where the Red agent succeeded against the LLM Blue, it typically had to work harder and longer. The average detection/containment time by Blue was around the midpoint of the attack chain. That is, Blue often caught the attack by the time Host A was under attack or just after Host A was compromised, which often forced Red to slow down and switch tactics. By contrast, baseline Blue is often only noticed at the final stage (if at all).

On offense, the LLM-driven Red agent was more effective than the baseline script in the sense that it found ways to succeed that the baseline attacker might not try. For instance, the LLM Red occasionally chose to send a phishing email to Host C's user to steal credentials as an alternative to exploiting Host A directly. In some trials, this social engineering approach bypassed Host A's hardened perimeter entirely (since Host C's user might reuse a password on Host B). The static baseline attacker had no such capability. However, those innovative strategies also created different patterns for Blue to potentially catch (e.g., an unusual login to Host B from Host C's account at an odd hour). We gave the Blue LLM awareness of typical user behavior, so in one case it did catch the phishing-based lateral move because the login was flagged as anomalous.

Red Adaptation and Arms-Race: We also experimented with allowing the Red agent to adapt when it started failing consistently. In those experiments, the contest became more protracted. For example, Red learned after a few failures that Host A's direct exploits were futile, so it switched entirely to stealthy methods like phishing or using Host A in a non-traditional way (instead of exploiting a vulnerability, it might trick an admin of Host A to run malicious code via social engineering). These new tactics would sometimes restore Red's chance of success, until Blue adapted to them as well. The net result was an arms race curve: success rate oscillated or leveled off at an equilibrium. In one scenario with mutual adaptation, the Red success rate stabilized around ~40% after many iterations, meaning Blue couldn't guarantee a stop, but Red couldn't get an easy win either—each run was a toss-up depending on whether Red's latest trick was something Blue had seen before or not. This closely mirrors real-world experiences where attackers and defenders continually evolve and neither side retains a permanent upper hand for long. It was encouraging that our simulation could capture this dynamic. It also underscores that continuous learning on both sides leads to the emergence of more and more sophisticated strategies, which is exactly the environment we want to train defenders in. Overall, these results indicate that replacing a static rule-based defender with an explainable LLM-driven agent can cut successful breaches roughly in half even without learning, and the adaptive version further drives the success rate down to single digits.

Scalability Study

We evaluate the same red–blue loop on emulations with 10/20/30 hosts by adding benign services and background traffic. We report (i) attack success probability, (ii) mean detection latency, and (iii) agent decision latency under different LLM backends. Results show the framework remains functional as environment size increases, while larger LLMs improve plan quality at the cost of higher latency.

4.4 Explainability and Agent Insights

One of the most rewarding aspects of the experiments was analyzing the explanations produced by the agents. They provided a rich narrative of the engagement, essentially an automatic play-by-play analysis that is rarely available in real red team exercises (where one has to do painstaking debriefs to reconstruct what happened).

Quality of Explanations: Generally, the explanations were accurate and informative. For the Red agent, the explanations usually correctly identified the reason a tactic was chosen. For example, a typical Red explanation: "I will try a SQL injection on Host A's web page because it might not be sanitized and could give me database access on Host B through Host A." In a case where that was indeed a vulnerability we seeded, it was spot on. There were a few cases of hallucination: e.g., Red saying "Host A appears to run an outdated OpenSSH, so I'll exploit it" when in fact we hadn't given any clue about the OpenSSH version. In those cases, the attempted action might fail, and we (as analysts) could see that the Red agent made an assumption. But interestingly, this is also something human attackers do—sometimes they try an exploit guessing a version. The difference is that the human knows they're guessing, whereas the LLM stated it confidently. This is a

minor discrepancy that didn't affect outcomes much because the environment simply treated it as a failed exploit (and Blue got an explanation of Red's intent which was educational in itself).

For the Blue agent, the explanations often read like a seasoned analyst's reasoning. A compelling example: at one point Blue isolated Host C with the explanation "Host C started scanning the network, which normal users shouldn't do. It's likely been compromised and used as a pivot, so I'm isolating it to contain the threat." That was exactly what had happened (Red had turned Host C into a scanning pivot after compromising it via phishing). The Blue agent connected the dots between a normally benign host and malicious behavior by understanding user roles and typical behavior, which is something we hoped the LLM could do with its world knowledge.

We also evaluated explainability by sharing some of the logged explanations with external security experts (three volunteers with professional SOC experience).

In addition to qualitative feedback, the experts provided structured ratings for each sampled XAI trace using the rubric described in Section 3.4. Table 3 summarizes the aggregated results across trials, including normalized trust scores and expert-alignment percentages.

They were asked to review a timeline of actions and explanations (without knowing whether it was an AI or human who produced them) and give feedback. They consistently commented that the rationale given was generally sound. In fact, one expert thought these were write-ups from a skilled red team operator explaining their steps in a tabletop exercise! When we revealed it was AI-generated, they were surprised by the level of detail and coherence. There were a few critiques: occasionally, an explanation was too certain about an assumption (like the Blue agent saying "definitely compromised" when it was just highly likely), which could be toned down. But overall, they felt the explanations made the scenario easy to follow and could help a human team learn from the exercise.

Case Study Walkthrough: To provide a concrete sense of how the explainable simulation plays out, here's a compressed walkthrough from one of our experiment runs (with learning enabled). We combine both agents' perspectives:

Step 1 (Red): Action: "Scan Host A for open ports." Explanation: "I want to see what services Host A is running to find a potential entry point."—Environment: Host A has ports 80 (HTTP) and 22 (SSH) open; Red learns this.

Step 2 (Blue): Action: "Enable heightened logging on Host A's firewall." Explanation: "Host A is internet-facing; after a port scan alert, I suspect it may be targeted, so I'll increase monitoring."—Environment: This doesn't thwart Red directly but will make Blue aware of subsequent port scans or connection attempts in detail.

Step 3 (Red): Action: "Exploit Host A's web service via known Apache Struts vulnerability." Explanation: "The scan suggests Host A is running Apache. If it's using a vulnerable Struts framework, I can execute commands on the server."—Environment: Suppose Host A indeed has that flaw; the exploit succeeds, Red gets a foothold (remote code execution on Host A).

Step 4 (Blue): Blue's observation includes an alert (the web service crashed or behaved oddly). Action: "Investigate Host A for signs of compromise (check processes and web server logs)." Explanation: "Host A's web service is behaving abnormally (possibly crashed/restarted). This could indicate an exploit. I will inspect running processes and recent logs on Host A."—Environment: Blue finds an unfamiliar process running with Apache's privileges—a likely webshell.

Step 5 (Blue, same turn): Action: "Isolate Host A immediately." Explanation: "Confirmed suspicion: Host A appears compromised (unexpected process found). I will isolate it to prevent attacker from going further."—Environment: Host A's network connections are cut off by Blue.

Step 6 (Red): Red's observation now is that connection to Host B from Host A fails (if it tried) and Host A commands aren't responding externally. Red infers Host A might be contained. Action: "Attempt a phishing attack on Host C's user to regain access." Explanation: "Host A is no longer accessible (likely detected). I will try another route: send a crafted phishing email to an internal user (Host C) to get credentials or access inside."—Environment: Suppose Red successfully phishes Host C's user (gets their Windows credentials).

Step 7 (Blue): Perhaps an alert triggers on Host C (antivirus flags the phishing payload or unusual login timing). Action: "Reset User C's domain password and scan Host C." Explanation: "We got an alert that Host C's user executed a suspicious file. I suspect a phishing attack. I'll cut off that user's access by resetting password and scan Host C for malware."—Environment: This likely boots Red out if they were using that credential live, and the scan might remove the malware. Host C is now under scrutiny.

Step 8 (Red): Red finds itself without a foothold again (Host A isolated, Host C credential lost). It might try another approach or consider the attempt failed. In this run, let's say time's up or Red's options are exhausted, so the scenario ends with Blue having successfully defended Host B.

This suggests that the explainable simulation not only improves security outcomes but also serves as a rich narrative resource for training and auditing human security teams.

4.5 Comparison with Prior Approaches

Although our evaluation primarily focused on comparing against a basic baseline, it's informative to contextualize our results with prior approaches in the literature:

Compared to traditional red team exercises (human-led), our autonomous framework can run much more frequently and consistently. It won't match human creativity in all aspects (especially social engineering nuances), but interestingly, the LLM Red agent did demonstrate creativity within its domain knowledge. The key benefit is that, unlike a one-off human test, our system can be left running continuously, providing constant vigilance. The explainable output also preserves the knowledge that in human exercises might remain tacit or only captured in a final report.

Compared to automated penetration testing tools (which often use scripted sequences or simple ML to pick attacks), our LLM-based Red is more flexible and context-aware. Tools like Core Impact or OpenVAS follow known patterns—they might find known CVEs and exploit them, but an LLM can think "outside the script" (for example, trying a phishing approach when technical exploits fail). This makes the red side of our simulation more formidable and closer to a thinking human adversary.

Compared to machine learning intrusion detection systems, our Blue agent is not just a classifier raising alerts, but an agent taking actions and reasoning. Many modern IDS/IPS use ML to flag anomalies, but then rely on humans to respond. Our Blue agent bridges that gap by interpreting anomalies and responding autonomously (and explaining its response). In terms of detection capability, a trained ML model might outperform an LLM on very specific pattern recognition (like spotting byte-level malware signatures), but the LLM's advantage is broad understanding—it can catch things the ML wasn't explicitly trained on by reasoning ("this process has a suspicious name and was spawned by a web server, that's fishy").

Compared to reinforcement learning (RL) agents in research (such as those trained in CyberBattleSim or similar environments), our LLM agents achieved strong performance with far less training data and with built-in explainability. RL agents usually need thousands of episodes to learn to defend a network and they often learn narrow policies that might fail if the scenario changes even slightly. Our LLM Blue started with a wealth of general knowledge (like "scans are bad", "failed logins might indicate brute force"), which gave it a head start. It improved with just a handful of runs because it could integrate new facts into its reasoning rather than needing to tweak a million weight parameters. Also, RL policies are essentially black boxes—one would

have to analyze reward curves or do state/action mapping to interpret them, whereas our LLM explicitly tells us why it does something, every time. This is a significant qualitative advantage when deploying in real environments where trust and verification are crucial [14].

In terms of raw effectiveness, if one had a perfect static rule-based system tailored to the scenario, that could potentially block everything, but it would be brittle (a new tactic goes unseen). Our approach is adaptive and robust to changes, which is a more realistic requirement given how quickly new threats emerge. The trade-off is that an LLM could theoretically make a mistake or be manipulated in ways a simple system wouldn't, so it introduces a new surface of AI-related risks (e.g., prompt manipulation by an attacker, which is an interesting avenue for future exploration: could a clever Red attempt to "fool" the Blue's LLM via crafted log entries?). But overall, we believe the ability to reason and explain outweighs those risks when mitigated properly [11].

Limitations of Baselines. Our quantitative experiments focus on a handcrafted script-based attacker and a static rule-based defender as baselines. We do not yet perform a head-to-head empirical comparison against state-of-the-art automated breach-and-attack simulation products or reinforcement-learning agents such as those provided by CyberBattleSim. While this limits external validity, it allows us to cleanly isolate the impact of GenAI-driven reasoning and explainability. A systematic benchmark against industrial BAS platforms and RL-based defenders is left for future work.

Having evaluated and discussed these results, we next summarize our contributions and consider the broader implications, as well as promising directions for future research building on this work.

Threats to Validity and Data Bias. Our evaluation has several limitations. First, the LLM agents are based on a general-purpose model whose pre-training data are not fully known; we cannot rule out the possibility that some attack or defense patterns in our scenarios resemble examples seen during training. To mitigate this risk, all experiments use synthetic network topologies and hand-crafted attack chains rather than logs from real organizations. Second, the enterprise network we simulate is intentionally small, which simplifies reasoning but may under-represent the complexity and noise of large-scale production environments. Finally, our user study on explanation quality involves a small number of security experts, so the reported trust scores should be interpreted qualitatively rather than as statistically definitive. Future work will extend the evaluation to more diverse datasets, larger networks, and broader analyst populations.

5 Conclusion

In this work, we presented a novel framework for autonomous cyber offense and defense using generative AI, specifically explainable LLM agents. We demonstrated that an LLM-powered Red Team can emulate sophisticated attacker behavior, while an LLM-powered Blue Team can effectively detect and counter attacks, especially when enhanced with an adaptive learning loop. Our simulation experiments showed that the GenAI-driven Blue agent, with its ability to reason and learn, can dramatically improve a network's resilience over successive attack iterations. Equally important, our approach maintains transparency by providing natural language explanations for each action, which is crucial for building trust in AI decisions within the cybersecurity domain [15].

Implications: The development of explainable autonomous red and blue agents opens up exciting opportunities for both research and practical security management. For defenders, such a system can serve as an ever-vigilant "cyber sparring partner"—continuously testing defenses with new attacks and suggesting improvements. This goes beyond periodic penetration testing; it creates a continuous feedback loop of hardening. The explainability means that this is not a black-box adversarial ML tool, but one whose insights can be directly understood and applied by human analysts (for example, learning about a new attack

technique the AI used, or trusting the AI's defensive action rationale in a high-pressure incident). For the broader security community, our work underscores the viability of using advanced AI (LLMs) not just for automating isolated tasks (like malware classification or log analysis), but for orchestrating complex, dynamic cyber operations autonomously.

However, our study also highlights key considerations. AI is a double-edged sword: the same LLM that empowers a defender can be wielded by attackers. Our Red agent is essentially a benign use of what a malicious actor might deploy. This reinforces the urgent need for defenses to incorporate AI—to fight fire with fire—but to do so with safeguards. Explainability is one such safeguard, ensuring humans remain in the loop and can intervene or correct course when the AI behaves in undesired ways.

6 Future Work

Real-world Integration: The current framework is a simulation. A logical next step is integrating an LLM Blue agent into a real network environment in a monitoring/advisory capacity. For example, it could interface with a Security Information and Event Management (SIEM) system or an orchestration platform, taking actions (with human approval as needed) on actual infrastructure. This requires addressing reliability and safety—ensuring the agent's actions won't disrupt normal operations or can be overridden easily if mistaken.

Advanced Attack Strategies: Our Red agent, while intelligent, can be made even more formidable by incorporating more advanced tactics (e.g., stealthy malware that avoids detection, or adversarial inputs designed to confuse AI-based detectors). Investigating how the Blue agent fares against such advanced threats, and possibly how the Blue agent's own LLM can be hardened against prompt or input manipulation by adversaries, will be critical. This relates to research in adversarial machine learning and AI robustness.

Scaling and Complexity: Future work should explore scaling the simulation to larger and more complex network environments (including cloud infrastructure, IoT devices, etc.). This likely entails using hierarchical models or dividing tasks (for example, separate LLM agents for network monitoring vs. host-based actions, under a coordinating logic). Efficient summarization of large state spaces for the LLM's consumption will be an important aspect [12].

Human-AI Collaboration: We envision a hybrid model where human security teams work alongside AI agents. Future research could explore interfaces that allow a human analyst to query the AI agents (e.g., "Why did you block that IP?" or "What is the attacker likely to do next?") and get coherent answers, leveraging the explainability component. Conversely, humans could provide feedback or high-level directives to the agents ("focus on protecting server X", "do not shut down any system without asking me"), creating a supervised autonomy that combines the best of both worlds.

Formal Validation and Policies: To deploy autonomous agents in production, organizations will need trust that the AI will behave within acceptable bounds. Developing formal safety constraints or rule-based wrappers around the LLM decisions could provide guarantees (e.g., a rule that the Blue agent will never disable a critical service unless a human approves, regardless of LLM output). Integrating such hard constraints with the flexible LLM reasoning is an interesting technical challenge.

Improving Explainability: While our current agents already explain their actions, future work might enhance this by generating more comprehensive reports or visual aids. For example, automatically mapping the attack path taken onto a network diagram, or highlighting in an event timeline which events led the Blue agent to suspect a breach. Another direction is to incorporate counterfactual explanations (e.g., Blue agent could explain what evidence would have made it take a different action, which can be insightful).

User Studies and Training: As our framework can serve as a training simulator for cybersecurity teams, conducting formal user studies would be valuable. This would assess how interacting with or observing the

AI agents impacts human learning and preparedness. Does the explainable AI simulation improve junior analysts'skills faster than traditional training? Such studies can quantify the educational benefit and guide further development of the system as a teaching tool.

Finally, we summarize the actionable future research roadmap—including scaling the testbed, strengthening RAG, and improving robustness—in Table 9.

Table 9: Simplified roadmap for future research and development in GenAI-Powered cyber defense.

Theme	Primary Objective	Key Tasks	Expected Deliverables
Real-world Integration	Deploy Blue LLM agent in real network environments with human oversight	Connect to SIEM/SOAR systems, test advisory actions, ensure rollback safety	Integration playbooks, operation manual
Advanced Attack Strategies	Strengthen Red agent with adversarial and stealth capabilities	Add adversarial samples, simulate prompt injection and evasion	Robustness evaluation report, adversarial dataset
Scaling & Complexity	Expand simulation to multi-domain, cloud, and IoT environments	Build hierarchical agents, optimize state summarization	Scalable architecture model, multi-domain scenario library
Human-AI Collaboration	Enable cooperative defense between analysts and AI	Design interactive interfaces, add human-in-the-loop commands	Analyst-AI dashboard, collaboration protocols
Formal Validation & Policies	Ensure AI safety and rule compliance	Add formal policy layer, validate through replay testing	Verified policy set, compliance checklist
Improving Explainability	Enhance clarity and visual reasoning	Generate attack graphs, timelines, counterfactual reports	Visual analytics reports, explanation toolkit
User Studies & Training	Use simulation as an educational platform	Conduct A/B user studies, measure learning outcomes	Training syllabus, skill evaluation report

In conclusion, GenAI-powered autonomous cyber offense-defense appears to be a promising paradigm for enhancing security. By combining the creativity and knowledge of advanced AI with the vigilance of automated simulation, organizations can continuously assess and bolster their defenses. Our explainable LLM red-vs.-blue framework is a step toward resilient, intelligent cyber defense systems that not only fight back against attackers in real-time but also transparently communicate their reasoning. We hope this work inspires further research at the intersection of cybersecurity, artificial intelligence, and explainability, driving us closer to a future where AI serves as a reliable guardian of digital infrastructure rather than a mysterious black-box.

Acknowledgement: The author would like to express sincere gratitude to all colleagues and mentors who provided valuable discussions and constructive feedback during the development of this research. Special thanks are extended to the cybersecurity research community for inspiring the conceptual design of autonomous Red-Blue simulations and for fostering an open environment for innovation in AI-driven cyber defense. The computational resources and academic support that facilitated this study are also gratefully acknowledged.

Funding Statement: The author received no specific funding for this study.

Availability of Data and Materials: The simulation code, configuration files, and anonymized logs that support the findings of this study will be released in an open-source repository upon publication. During the review phase, these materials are available from the corresponding author on reasonable request. No proprietary or sensitive real-world data were used.

Ethics Approval: This study does not involve human participants, personal data, or any real-world network intrusion testing. All experiments were conducted within a controlled simulation environment in compliance with standard ethical guidelines for cybersecurity research. No harmful actions were executed on any production systems or external networks.

Conflicts of Interest: The author declares no conflicts of interest.

References

1. Lai YC, Motter AE, Nishikawa T. Attacks and cascades in complex networks. Lect Notes Phys. 2004;650(650):299–310. doi:10.1007/978-3-540-44485-5_14.
2. Elouafiq A, Khobalatte A, Benhallam W, Iraqi O, Rachidi TE. Aggressive and intelligent self-defensive network: towards a new generation of semi-autonomous networks. In: Proceedings of the International Conference on Networked Digital Technologies; 2012 Apr 24–26; Dubai, United Arab Emirates. Berlin/Heidelberg, Germany: Springer; 2012. p. 344–54.
3. Waizel G. Bridging the AI divide: the evolving arms race between AI-driven cyber attacks and AI-powered cybersecurity defenses. Int Conf Mach Intell Secur Smart Cities (TRUST) Proc. 2024;1:141–56.
4. Wu W, Tian Z, Sun Y, Li Y, Cui Y. Study on the application of dynamic continuous trust assessment model based on clustering mechanism in network security optimization. J Cyber Secur Mobil. 2025;14(1):47–74. doi:10.13052/jcsm2245-1439.1413.
5. Balazinska M, Castro P. Characterizing mobility and network usage in a corporate wireless local-area network. In: Proceedings of the 1st International Conference on Mobile Systems, Applications and Services; 2003 May 5–8; San Francisco, CA, USA. p. 303–16. doi:10.1145/1066116.1066127.
6. Whitman J, El-Karim A, Nandakumar P, Ortega F, Zheng L. Self-learning algorithms in cyber defense systems. [cited 2025 Jan 1]. Available from: www.researchgate.net/publication/391857637_Self-Learning_Algorithms_in_Cyber_Defense_Systems.
7. Banerjee A, Mishra A, Dana SK, Hens C, Kapitaniak T, Kurths J, et al. Predicting the data structure prior to extreme events from passive observables using echo state network. Front Appl Math Stat. 2022;8:955044. doi:10.3389/fams.2022.955044.
8. Wang J, Li W, Wang Y, Tao R, Du Q. Representation-enhanced status replay network for multisource remote-sensing image classification. IEEE Trans Neural Netw Learning Syst. 2024;35(11):15346–58. doi:10.1109/tnnls.2023.3286422.
9. Meddeb R, Jemili F, Triki B, Korbaa O. A deep learning-based intrusion detection approach for mobile *Ad-hoc* network. Soft Comput. 2023;27(14):9425–39. doi:10.1007/s00500-023-08324-4.
10. Boddu Y, Manimaran A. Optimizing multivariate time series forecasting with LSTM: a hybrid scaling and layer normalization framework utilizing logistic and sigmoid-curve transformations for enhanced predictive accuracy. Comput Econ. 2025;1–43. doi:10.1007/s10614-025-11103-y.

11. Zanotti G, Chiffi D, Schiaffonati V. AI-related risk: an epistemological approach. Philos Technol. 2024;37(2):66. doi:10.1007/s13347-024-00755-7.
12. Almutairi R, Bergami G, Morgan G. Advancements and challenges in IoT simulators: a comprehensive review. Sensors. 2024;24(5):1511. doi:10.3390/s24051511.
13. Fazlollahi AM, Yilmaz R, Winkler-Schwartz A, Mirchi N, Ledwos N, Bakhaidar M, et al. AI in surgical curriculum design and unintended outcomes for technical competencies in simulation training. JAMA Netw Open. 2023;6(9):e2334658. doi:10.1001/jamanetworkopen.2023.34658.
14. Kim SSY, Watkins EA, Russakovsky O, Fong R, Monroy-Hernández A. Humans, AI, and context: understanding end-users' trust in a real-world computer vision application. In: Proceedings of the 2023 ACM Conference on Fairness, Accountability, and Transparency; 2023 Jun 12–15; Chicago, IL, USA. p. 77–88.
15. Xu M, Fan J, Huang X, Zhou C, Kang J, Niyato D, et al. Forewarned is forearmed: a survey on large language model-based agents in autonomous cyberattacks. arXiv:2505.12786. 2025.