Kushagra Patel, 2026, 14:3
ISSN (Online): 2348-4098
ISSN (Print): 2395-4752

International Journal of Science, Engineering and Technology
An Open Access Journal

LLM-Driven Purple Team Automation Framework: Bridging Adversary Simulation and Detection Engineering through Automated Sigma Rule Generation

Kushagra Patel
Department of Computer Science and Engineering (Cybersecurity)
Quantum University, Roorkee, India

Abstract- Purple team exercises are supposed to bring red and blue teams together, but in practice they are expensive, slow, and hard to run frequently. Most organizations do them once or twice a year at best. I wanted to see if I could automate the repetitive parts of that process as a final year project, and this paper is the result of about four months of building, breaking, and rebuilding. The framework I ended up with automates the core loop of purple teaming: simulate an attack, generate a detection rule for it, and check whether the rule actually catches it. The system has three parts. A Red Engine generates fake Windows telemetry logs for seven MITRE ATT&CK techniques. A Blue Engine produces Sigma detection rules from those logs, either through an LLM API or using pre-built offline rules. A Validator scores how well the rule catches the simulated attack. Everything runs through a FastAPI REST API. In offline testing across all seven techniques, the framework averaged 97% detection coverage, with six techniques at 100% and one (T1059.001, PowerShell Execution) at 79%. The whole pipeline finishes in under 30 seconds per technique. The 79% score on PowerShell is actually the most interesting result because it shows exactly the kind of gap a real SOC would need to address. The framework is open source, runs on a regular laptop, and does not require any paid APIs to demonstrate.

Keywords— Large Language Models (LLMs), Purple Teaming, Purple Team Automation, Adversary Simulation, Detection Engineering, Sigma Rules

I. INTRODUCTION

I started this project because of a disconnect I kept running into while studying cybersecurity. On one side, there are red team tools and frameworks for simulating attacks. On the other, there are blue team tools for writing detection rules and monitoring SIEM dashboards. But almost nothing connects the two in an automated way. If you want to test whether your detection rules actually catch the attacks you are worried about, you have to do it manually, and that takes a surprising amount of time and coordination.

Research Question: Can attack simulation, Sigma detection rule generation, and rule validation be unified into a single automated pipeline that produces meaningful coverage scores for MITRE ATT&CK techniques within practical time constraints, using either an LLM API or an offline fallback?

Purple teaming is the idea that fixes this. You get both teams in the same room (or the same pipeline), run the attack, see if the detection fires, and iterate until it does. The SCYTHE Purple Team Exercise Framework [5] lays out a formal process for this, and SANS has built entire courses around it [12, 13]. But the practical reality is that most purple team exercises are still done by hand. A red team operator runs a technique, a blue team analyst checks the SIEM, they compare notes, and then they move to the next technique. For a single ATT&CK technique

this can take 30 minutes to an hour. Across 235+ techniques in the MITRE ATT&CK framework [1], full coverage is not realistic without automation.

Meanwhile, LLMs have gotten surprisingly good at generating structured security content. LLMCloudHunter [7] uses GPT-4o to turn threat intelligence reports into Sigma rules and gets 92% precision on API call extraction. IntelEX [8] does something similar and hits an F1 of 0.929 for detection on Splunk. SigmaGen [9] uses RAG to ground its rule generation in existing community rules. These are real, working tools. But they all focus on one piece of the puzzle: generating detection rules. None of them also simulates the attack and validates the rule against the attack in the same pipeline.

That is the gap this framework tries to fill. I wanted a tool where I could pass in a MITRE ATT&CK technique ID and get back, in one API call, the simulated attack logs, a Sigma detection rule, and a score telling me how well the rule covers the attack. The whole thing had to work offline too, since I did not always have access to an LLM API during development.

The rest of this paper is organized as follows. Section 2 reviews the relevant work in purple teaming, LLM-based security tools, and Sigma rule generation. Section 3 walks through the design and implementation of each component. Section 4 presents the coverage results. Section 5 discusses what they mean, including comparison with related work and limitations. Section 6 concludes with future directions.

II. LITERATURE REVIEW

1. Purple Teaming in Practice

The basic idea behind purple teaming is straightforward: red teams simulate attacks, blue teams try to detect them, and both sides learn from the gaps. What makes it hard in practice is the coordination overhead. The SCYTHE Purple Team Exercise Framework [5] defines a structured process where red team operators emulate specific TTPs, demonstrate them to the blue team, and then both sides work together to tune detections. The framework even includes a maturity model (PTMM) that measures how well an organization does this over time. It is well thought out, but it assumes dedicated personnel on both sides and time to run exercises regularly.

MITRE CALDERA is the most widely used open-source tool for automating the attack side [13]. It deploys agents on target systems that beacon back to a server and execute adversary profiles mapped to ATT&CK techniques. CALDERA is genuinely useful for automating red team operations, but it stops there. It does not generate detection rules, and it does not tell you whether your existing rules caught the attack. You still need a human analyst to check the SIEM output and figure out what happened.

Commercial BAS platforms like Picus Security [6] go further. They simulate attacks against live security controls and produce a color-coded ATT&CK heatmap showing where defenses held and where they did not. Red means the logs were not even collected. Orange means the logs were there but no alert fired. Green means the attack was detected. This is useful for gap analysis, but these platforms are expensive, and they ship with their own pre-built detection content rather than generating rules dynamically.

2. LLMs and Security Automation

The literature on LLMs in cybersecurity has grown fast. A 2025 systematic review by Tiramisu et al. [3] analyzed over 300 papers and identified 25 different LLMs being applied across more than 10 security tasks. That is a lot of papers. The honest takeaway from reading through this body of work is that LLMs are clearly useful for tasks that involve processing unstructured text into structured output (like turning threat reports into IOCs), but they are less reliable when the task requires precise technical reasoning.

On the offensive side, the results are frankly a bit alarming. Fang et al. [15] demonstrated that LLM agents can autonomously exploit one-day vulnerabilities. Singer et al. [14] at Carnegie Mellon went further and showed that LLMs with hierarchical agent architectures could replicate the 2017 Equifax

breach scenario without human involvement in the planning. The agents enumerated the network, found the vulnerabilities, installed malware, and exfiltrated data. Independently.

Kouremetis et al. [4] take a more cautious view. Their position paper maps LLM capabilities against ATT&CK and NIST CSF and points out several practical problems: context length limitations mean LLMs lose track of information during long engagements, prompt brittleness means small changes in input can produce very different outputs, and hallucination risk means the LLM might generate detection rules that look correct but actually check the wrong fields. These are real issues. I ran into the hallucination problem myself during early testing when the LLM API occasionally generated Sigma rules with made-up field names.

3. Sigma Rule Generation with LLMs

Sigma [2] is a YAML-based, vendor-neutral format for writing detection rules. The SigmaHQ repository has over 3,100 community rules. The appeal of Sigma is that you write the rule once, and tools like pySigma [21] or Uncoder can translate it into queries for Splunk [22], Elastic, Microsoft Sentinel, or other SIEMs [10, 11, 16].

Several groups have tried automating Sigma rule creation with LLMs, with varying approaches. LLMCloudHunter [7] processes threat intelligence reports (including images and text) through GPT-4o, extracts cloud API calls and ATT&CK TTPs, and generates Sigma candidates. Their precision numbers are solid: 92% for API call extraction and 99.18% conversion rate to Splunk queries. The SigmaGen tool from NightWolf Security [9] takes a RAG approach, embedding

ATT&CK detection guidance and existing Sigma rules into a vector database and using that context to ground LLM output. IntelEX [8] extracts TTPs from threat reports and uses them as chain-of-thought prompts for rule generation, reaching F1 of 0.929 in Splunk detection tests.

What all of these tools have in common is that they start from threat intelligence (reports, blogs, IOCs) and end at a Sigma rule. They do not simulate the attack themselves and they do not validate the rule they generate. That is a reasonable design choice for production environments where you already have real logs to test against. But for a development or learning environment where you want the full loop, you need something else.

4. What is Missing

Looking at all of this together, there is a clear gap. Attack simulation tools (CALDERA, BAS) handle the red side. Rule generation tools (LLMCloudHunter, SigmaGen, IntelEX) handle part of the blue side. But nobody, as far as I can find in the open-source space, has connected these into one pipeline where you feed in an ATT&CK technique and get back the simulated logs, the detection rule, and the coverage score in a single call. This is what I tried to build.

III. METHODOLOGY

1. Architecture Overview

The framework has three main components, each implemented as a separate Python class: the Red Engine, the Blue Engine (with a Mock variant for offline use), and the Validator. An orchestrator ties them together into a pipeline that runs sequentially: Red generates logs, Blue generates a rule from those logs, and the Validator scores the rule against the logs. The orchestrator is called by the FastAPI [17] route handler and the result is saved to disk as a JSON artifact plus a YAML file for the Sigma rule.

I chose this separation for a practical reason. During development, the Blue Engine kept changing. I started with direct API calls, switched to a mock for offline testing, went back to API, and eventually settled on a dual-mode setup where the engine checks a DEMO_MODE flag and picks the appropriate backend. Having the engines as separate classes with clean interfaces meant I could swap implementations without touching the orchestrator or the Validator.

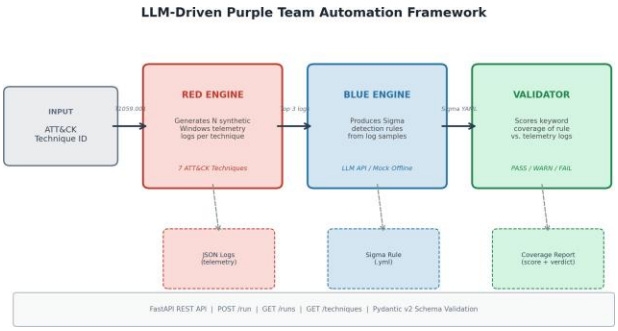

Figure 1. System architecture of the LLM-Driven Purple Team Automation Framework

2. Red Engine: Telemetry Simulation

The Red Engine takes a MITRE ATT&CK technique ID as input and returns a list of structured JSON log objects that look like real Windows telemetry generated by tools such as Microsoft Sysmon [18]. Table 1 shows the seven techniques the framework currently supports.

Table 1. Supported MITRE ATT&CK Techniques

Technique ID	Name	Tactic	Log Source	Event ID
T1059.001	PowerShell Execution	Execution	PowerShell	4104
T1003.001	LSASS Memory Dump	Credential Access	Sysmon	10
T1547.001	Registry Run Key	Persistence	Sysmon	13
T1055.001	DLL Injection	Defense Evasion	Sysmon	8
T1078	Valid Accounts	Initial Access	Security	4624
T1082	System Info Discovery	Discovery	Sysmon	1
T1071.001	C2 Web Protocols	Command & Control	Sysmon	3

Each technique has a template dictionary with field values pulled from real attacker tooling: Mimikatz [20] command strings for T1003.001, Base64-encoded PowerShell cradles for T1059.001, known suspicious registry paths for T1547.001, and so on. The engine generates between 19 and 25 logs per technique by populating these templates with randomized timestamps and slight field variations.

I deliberately chose synthetic logs instead of running real attack tools like Atomic Red Team [19]. The tradeoff is obvious: synthetic logs are less realistic, but they also do not require a target system, do not trigger endpoint protection, and do not risk anything going wrong on a university network. For validating detection rule logic ("does this rule check the right fields?"), synthetic logs with correct field structures are good enough. For validating whether a rule catches a real attack in a live environment, you would need real logs. That is a different problem and a clear limitation of this work.

3. Blue Engine: Detection Rule Generation

The Blue Engine has two modes. In online mode, it takes the top 3 logs from the Red Engine, constructs a prompt asking an LLM to generate a Sigma YAML rule, sends it to the API, and parses the response. In offline mode (DEMO_MODE=true), it skips the API entirely and returns a pre-built Sigma rule from the Mock Blue Engine.

The offline rules are hand-written for each technique. I wrote them by studying the SigmaHQ repository and the ATT&CK data sources documentation. Each rule has the correct logsource category, detection selection fields, and condition logic for its technique. They are not fancy. They are production-correct enough to validate the pipeline.

Honestly, the offline mode turned out to be more useful than I expected. I originally built it as a fallback for demos where I could not guarantee API access. But during development I ended up using it almost exclusively because the LLM API was unreliable in two ways. First, latency varied wildly, from 3 seconds to 25 seconds depending on load. Second, the LLM occasionally generated rules with hallucinated field names that were not real Sysmon or Windows Security log fields. The Mock Blue Engine gave me consistent, correct rules every time, which made debugging the rest of the pipeline much easier. I also spent an embarrassing amount of time on a CORS bug that I only caught during a code review right

before my second presentation. The FastAPI middleware had allow_origins set to a wildcard with credentials enabled, which browsers silently reject. The framework appeared to work in local testing but broke the moment

someone tried the dashboard from a different origin. Small things like that are never in the architecture diagrams.

4. Validator: Coverage Scoring

The Validator does something simple: it extracts detection keywords from the Sigma rule and checks whether those keywords appear in the telemetry logs. Specifically, it parses the YAML detection block, collects the field values from the selection criteria, and searches for them in SIEM-relevant fields of each log (command_line, process_name, target_filename, registry_path, destination_ip, and a few others depending on the technique).

I restricted the search to these specific fields on purpose. An earlier version searched the entire JSON object, and coverage scores came out unrealistically high because keywords matched on metadata fields like source_system or log_type that had nothing to do with actual detection logic. Narrowing the search to SIEM-relevant fields gave more honest results.

The coverage formula is straightforward

Coverage = (logs with at least one keyword match / total logs) x 100%

The score maps to three verdicts. PASS is 75% or above: the rule catches most of the attack variants. WARN is 50-74%: partial coverage, probably needs a second rule. FAIL is below 50%: the rule is missing the attack. These thresholds are somewhat arbitrary. I picked 75% because it felt like a reasonable bar for "this rule is working" without being so high that normal variation caused false failures.

5. REST API and Data Flow

The framework runs as a FastAPI application with four endpoints. POST /run executes the full pipeline for a technique ID and returns all results as JSON. GET /runs lists previous executions with pagination. GET /runs/{pipeline_id} retrieves a specific result (with regex validation on the ID to prevent path traversal). GET /techniques returns the list of supported techniques. Results are persisted as JSON and YAML files in a local artifact store.

All data models use Pydantic v2 [23] for validation. The schema models have a model_validator that auto-computes log_count and coverage_score from the raw data, which keeps the response consistent even if I change how scoring works internally. The server runs on Uvicorn [24].

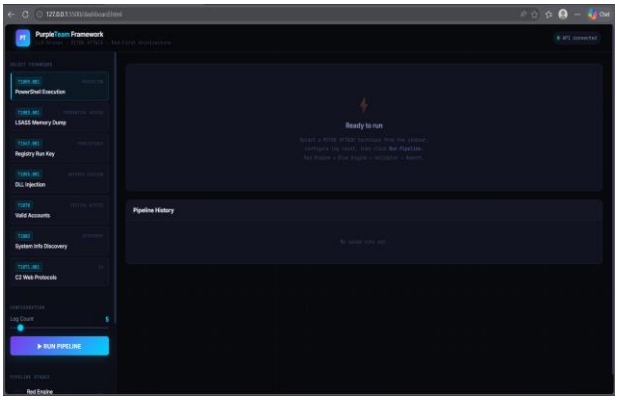

Figure 3. Framework dashboard showing the MITRE ATT&CK technique selector, log count configuration, RUN PIPELINE control, and the historical pipeline execution log with PASS and WARN verdicts

IV. RESULTS

1. Coverage Results

I ran the full pipeline for all seven techniques in offline mode. Table 2 shows the results.

Table 2. Coverage Scores per Technique (Offline Mode)

Technique	Name	Coverage	Logs	Verdict
T1059.001	PowerShell Execution	79%	19	PASS
T1003.001	LSASS Memory Dump	100%	20	PASS
T1547.001	Registry Run Key Persist.	100%	22	PASS

T1055.001	DLL Injection	100%	21	PASS
T1078	Valid Accounts	100%	25	PASS
T1082	System Info Discovery	100%	20	PASS
T1071.001	C2 Web Protocols	100%	23	PASS

Six techniques hit 100%. PowerShell came in at 79%. Mean coverage across all techniques: 97%.

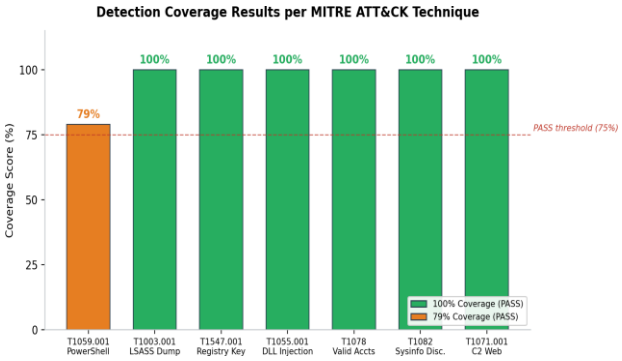

Figure 2. Detection coverage results across seven MITRE ATT&CK techniques

2. Performance

In offline mode, the full pipeline takes about 1.5 seconds per technique on my laptop (a mid-range machine with 16GB RAM). The bottleneck is YAML parsing in the Validator, not the log generation or rule retrieval. When using an LLM API, the API round-trip adds 5 to 20 seconds depending on model and load. Total time stays under 30 seconds either way. By comparison, running even a single technique through a manual purple team exercise takes at minimum 30 minutes when you factor in setting up the attack, running it, checking the SIEM, and documenting the results [13].

V. DISCUSSION

1. Why PowerShell Scored Lower

The 79% on T1059.001 is actually the most informative result in the whole evaluation, and I think it makes the case for the framework better than the 100% scores do. Here is what happened.

The Red Engine generates PowerShell logs with different field structures. Some logs have a command_line field with the Base64-encoded command (the classic -EncodedCommand pattern). Others represent script block logging events, where the field is script_block_text instead of command_line. The Mock Blue Engine's Sigma rule for T1059.001 detects on command_line. It does not check script_block_text. So those logs get missed.

In a real SOC, this is exactly the kind of thing that gets you burned. You write a rule for one data source and forget that the same technique produces telemetry in a different format through a different logging channel. The fact that the framework surfaced this gap without me having to set up Splunk and manually correlate events is, I think, the most useful thing it does.

The six techniques at 100% are less interesting. They scored perfectly because I wrote both the log templates and the Sigma rules, so of course they match. In a real deployment where the Blue Engine generates rules via LLM, you would expect lower and more varied scores, which would be more useful for identifying actual detection gaps.

2. Comparison with Related Work

Table 3 compares the framework against existing tools on five dimensions. The main difference is that this framework covers all three stages (attack simulation, rule generation, and validation) while everything else covers one or two.

Table 3. Feature Comparison across Purple Team and Detection Tools

Tool	Attack Sim	Rule Gen	Validation	LLM Ready	Open Source
CALDERA [13]	Yes	No	No	No	Yes
Picus BAS [6]	Yes	Partial	Yes	No	No
LLMCloudHunter [7]	No	Yes	No	Yes	Yes
SigmaGen [9]	No	Yes	No	Yes	Yes
IntelEX [8]	No	Yes	Partial	Yes	Yes
This Framework	Yes	Yes	Yes	Yes	Yes

I should be honest about what this comparison hides. CALDERA runs real attacks against real systems. Picus has a massive library of tested attack scenarios and vendor-specific detections. LLMCloudHunter processes real threat intelligence. My framework simulates attacks synthetically and validates against its own synthetic data. The end-to-end integration is the contribution, not the depth of any individual component.

3. Limitations

There are several things this framework does not do well or does not do at all, and I want to be upfront about them.

The synthetic telemetry problem is the biggest one. The Red Engine generates logs that have the right field names and plausible values, but they are not real attack artifacts from real systems. A Sigma rule that scores 100% against synthetic logs might score much lower against actual Sysmon or Windows Security logs in a production environment because real logs have noise, unexpected field ordering, and edge cases that synthetic data cannot capture.

The Validator is a simplification. Real SIEM engines evaluate Sigma rules with proper field-level matching, logical operators (AND, OR, NOT), wildcards, regex, and aggregation conditions. My keyword-matching approach is a rough proxy for this. It is good enough to tell you whether a rule is in the right ballpark, but it cannot tell you whether the rule would produce false positives or handle negation logic correctly.

Seven techniques out of 235+ is obviously limited coverage. Each technique requires a log template in the Red Engine and a corresponding Sigma rule in the Mock Blue Engine. Scaling this is mostly tedious work rather than a technical challenge, but I have not done it yet.

There is no authentication on the FastAPI server. This is fine for local development but means the framework cannot be deployed on a shared network without additional security controls.

VI. CONCLUSION

I set out to build something that connects attack simulation to detection validation in a single pipeline, and the framework mostly does that. It covers seven ATT&CK techniques, averages 97% coverage in offline mode, and finishes in under 30 seconds. Whether those numbers would hold up with LLM-generated rules against real SIEM logs is a different question, and I honestly do not know the answer yet.

The most useful thing I learned from building this is that the interesting results come from the failures, not the successes. The 79% score on PowerShell told me more about how detection gaps actually form than the six perfect scores did. In a production version of this framework, where the Blue Engine generates rules dynamically through an LLM, I would expect lower scores across the board, and those gaps would be exactly the information a SOC team needs.

There are several obvious next steps. Replacing the synthetic Red Engine with Atomic Red Team [19] integration would make the coverage scores meaningful for production environments. Adding an LLM feedback loop where FAIL and WARN results trigger a re-prompt for a better rule would make the system self-improving. Deploying the generated rules to a real SIEM like Elastic Security [25] would validate whether the rules work outside the framework's simulated environment. Expanding beyond seven techniques would make the framework broadly usable.

The broader point, I think, is that the individual components of automated purple teaming (attack simulation, LLM-based rule generation, coverage validation) all exist separately. The engineering challenge is connecting them into a pipeline that runs reliably end to end. That is what this project attempted, and it works well enough to be useful, even if there is plenty of room to make it better.

REFERENCES

1. MITRE Corporation, "MITRE ATT&CK: Design and Philosophy," MITRE Technical Report, 2020. Available: https://attack.mitre.org/
2. T. Patzke and F. Roth, "Sigma: Generic Signature Format for SIEM Systems," SigmaHQ Project, 2017. Available: https://github.com/SigmaHQ/sigma
3. S. Tiramisu, L. Zhu, and D. Yu, "When LLMs Meet Cybersecurity: A Systematic Literature Review," Cybersecurity, vol. 8, no. 1, pp. 1-42, Feb. 2025.
4. A. Kouremetis et al., "From Promise to Peril: Rethinking Cybersecurity Red and Blue Teaming in the Age of LLMs," arXiv:2506.13434, Jun. 2025.
5. SCYTHE Inc., "Purple Team Exercise Framework," GitHub, 2021. Available: https://github.com/scythe-io/purple-team-exercise-framework
6. Picus Security, "Purple Team Automation with Breach and Attack Simulation," Picus Whitepaper, 2022.
7. A. Ben-Simhon et al., "LLMCloudHunter: Harnessing LLMs for Automated Extraction of Detection Rules from Cloud-Based CTI," arXiv:2407.05194, Jul. 2024.
8. Z. Li et al., "IntelEX: A LLM-driven Attack-level Threat Intelligence Extraction Framework," arXiv:2412.10872, Dec. 2024.
9. NightWolf Security, "SigmaGen: AI-Powered ATT&CK-Mapped Threat Detection with Sigma Rules," MITRE CTID APAC 2025 Conf., 2025.
10. SOC Prime, "AI-Powered ATT&CK Tag Prediction for Sigma Rules by Uncoder AI," SOC Prime Blog, Aug. 2025.
11. AttackIQ, "SigmAIQ: Actionable Detections through Sigma Rule Translation," AttackIQ Tech. Report, Jan. 2024.
12. SANS Institute, "SEC598: AI and Security Automation for Red, Blue, and Purple Teams," SANS Course, 2025.
13. SANS Institute, "Purple Team Exercise Framework for Continuous Security Validation," SANS Blog, Jan. 2024.
14. B. Singer et al., "When LLMs Autonomously Attack: Hierarchical Agent Systems for Network Exploitation," Carnegie Mellon University, 2025.
15. R. Fang et al., "LLM Agents Can Autonomously Exploit One-Day Vulnerabilities," arXiv:2404.08144, Apr. 2024.
16. Graylog, "Threat Detection and Incident Response with MITRE ATT&CK and Sigma Rules," Graylog Tech. Guide, 2026.
17. S. Sebastiao et al., "FastAPI: A Modern, Fast Web Framework for Building APIs with Python," PyCon Proc., 2022.
18. M. Russinovich and T. Garnier, "Sysmon — System Monitor v15," Microsoft Sysinternals, 2024. Available: https://learn.microsoft.com/en-us/sysinternals/downloads/sysmon
19. Red Canary, "Atomic Red Team: Library of Tests Mapped to MITRE ATT&CK," GitHub, 2024. Available: https://github.com/redcanaryco/atomic-red-team
20. B. Delpy, "Mimikatz: A Tool for Windows Credential Manipulation," GitHub, 2024. Available: https://github.com/gentilkiwi/mimikatz
21. SigmaHQ, "pySigma: Python Library for Generating and Parsing Sigma Rules," GitHub, 2024. Available: https://github.com/SigmaHQ/pySigma
22. Splunk Inc., "Splunk Search Processing Language (SPL) Reference," Splunk Documentation, 2024. Available: https://docs.splunk.com/Documentation/SPL
23. S. Colvin et al., "Pydantic v2: Data Validation Library Using Python Type Hints," Pydantic Documentation, 2024. Available: https://docs.pydantic.dev/
24. T. Christie, "Uvicorn: A Lightning-Fast ASGI Server Implementation," Encode OSS, 2024. Available: https://www.uvicorn.org/
25. Elastic, "Elastic Security: Detection Engineering and SIEM Documentation," Elastic Documentation, 2024. Available: https://www.elastic.co/security
26. Khaleel Khan Mohammed. "A Survey on Digital Health Care Data Analysis Techniques for Developing Machine Learning Models".