

Automating Detection Lifecycle for APT Hunting: From CTI-Derived Sigma Rules to Infrastructure-Aware Rule Variants

Sepehr Ghaffarzadegan

**A Thesis
in
The Department
of
Electrical and Computer Engineering**

**Presented in Partial Fulfillment of the Requirements
for the Degree of
Master of Applied Science (Electrical and Computer Engineering) at
Concordia University
Montréal, Québec, Canada**

April 2026

© Sepehr Ghaffarzadegan, 2026

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Sepehr Ghaffarzadegan**

Entitled: **Automating Detection Lifecycle for APT Hunting: From CTI-Derived
Sigma Rules to Infrastructure-Aware Rule Variants**

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Electrical and Computer Engineering)

complies with the regulations of this University and meets the accepted standards with respect to
originality and quality.

Signed by the Final Examining Committee:

Dr. Abdelwahab Hamou-Lhadj Chair

Dr. Carol Fung External Examiner

Dr. Jun cai Examiner

Dr. Chadi Assi, Dr. Mourad Debbabi Supervisor

Dr. Boubakr Nour Co-supervisor

Approved by

Dr. Abdelwahab Hamou-Lhadj, Chair
Department of Electrical and Computer Engineering

2026

Mourad Debbabi, Dean
Faculty of Engineering and Computer Science

Abstract

Automating Detection Lifecycle for APT Hunting: From CTI-Derived Sigma Rules to Infrastructure-Aware Rule Variants

Sepehr Ghaffarzadegan

Cyber threat intelligence (CTI) reports contain important information about adversary behavior, yet transforming this information into deployable detection logic remains a largely manual and time-consuming task. Security practitioners must read narrative reports thoroughly, identify relevant attack techniques, extract and understand the attacks, and translate them into detection rules compatible with security monitoring systems. At the same time, detection rules remain static even as adversaries evolve. Attackers frequently change their techniques, substitute tools, modify execution paths, or move across platforms in order to evade fixed detections. These challenges create a gap between intelligence production and operational detection capabilities. This thesis addresses this gap by proposing a method for transforming CTI into adaptive detection rules. First, we introduce **AUTOSigma**, an automated solution for transforming unstructured CTI reports into relevant Sigma rules. Rather than relying solely on language models, **AUTOSigma** leverages a structured knowledge base to enrich partial input, matches the enriched content against a repository of existing Sigma rules, and then employs an LLM-as-a-Judge mechanism to iteratively validate the rules. By combining knowledge-driven enrichment, template-based rule grounding, and a multi-stage solution, **AUTOSigma** enables accurate, context-aware, and relevant rule generation, while reducing the manual effort required from security practitioners. Second, we present **ARMS**, an automated rule mutation solution designed to extend detection coverage under adversarial variation. **ARMS** begins with an existing detection rule and generates infrastructure-aware behavioral variants. **ARMS** identifies techniques related to the original attack through a three-dimensional proximity space that captures structural relations, functional similarity, and environmental constraints. Only techniques

compatible with the target infrastructure are retained. Subsequently, ARMS uses a RAG-based implementation to generate a new set of deployable detection rule variants. By generating the set of infrastructure-aware rule variants, ARMS improves detection coverage when attackers alter their techniques. Together, AUTOSIGMA and ARMS form a unified detection lifecycle that connects threat intelligence interpretation, rule synthesis, behavioral indexing, rule mutation, and deployment feedback. This lifecycle aims to support a continuous detection process in which intelligence drives detection creation and automated reasoning expands coverage as threats evolve. The results show that the combination of structured knowledge with language model capabilities can significantly accelerate CTI operationalization and contribute to more adaptive and resilient detection strategies against evolving APT campaigns.

Acknowledgments

First and foremost, I would like to thank Dr. Chadi Assi for kindly accepting the responsibility of supervising me and for believing in me throughout this journey.

I would also like to express my sincere gratitude to my co-supervisor, Dr. Boubakr Nour, for his continuous support, guidance, and encouragement. His dedication in helping me refine my ideas and tackle challenges is deeply appreciated.

I would also like to express my heartfelt appreciation to Dr. Makan Pourzandi and Dr. Mourad Debbabi, whose mentorship has had a profound impact on both my academic path. Their invaluable comments, suggestions, guidance, and encouragement have been instrumental in shaping this work.

I would also like to thank my research team for making this journey easier and more enjoyable. I feel incredibly fortunate to have had the opportunity to work with such supportive colleagues, and I will always be grateful for their dedication, encouragement, and belief in my potential.

Finally, I would like to express my deepest gratitude to my family for their unwavering support and encouragement, despite the distance between us. Their love, patience, and belief in me have made this achievement possible.

Contents

List of Figures	ix
List of Tables	xi
1 Introduction	1
1.1 The Threat Landscape and Cyber Threat Intelligence	1
1.2 Rule-based detection and the role of Sigma	2
1.3 The Dual Bottleneck in Detection	2
1.3.1 Bottleneck 1: The Manual Gap in CTI-to-rule Conversion	2
1.3.2 Bottleneck 2: The Fragility of Static Rules Against Attack Variants	3
1.4 Contributions	4
1.5 Thesis organization	5
2 Background and Related Work	6
2.1 Threat hunting and detection engineering context	6
2.2 Cyber Threat Intelligence and CTI extraction tools	7
2.3 MITRE ATT&CK and behavioral indexing	8
2.4 Sigma and detection rule portability	9
2.5 Automated rule generation	9
2.6 Rule robustness, mutation, and optimization	10
2.7 Research gaps and motivations	11

3	AUTOSigma: Automated Sigma Rule Generation	13
3.1	Motivation and Problem Statement	13
3.1.1	Motivation	13
3.1.2	Problem Statement	14
3.1.3	Contribution and Advantages of AUTOSigma	15
3.2	Threat Model and Assumptions	16
3.3	Automated Sigma Rules Generation	17
3.3.1	System Overview	17
3.3.2	Contextual Analysis	19
3.3.3	Attack Understanding	20
3.3.4	Template Matching	23
3.3.5	Rule Generation	25
3.4	Implementation	28
3.4.1	LLM Models	28
3.4.2	Datasets	29
3.5	Evaluation	29
3.6	Discussion	40
4	ARMS: Automated Rule Mutation Solution	41
4.1	Motivation and Problem Statement	41
4.1.1	Contribution and Advantages of ARMS	43
4.2	Threat Model and Assumptions	44
4.3	ARMS: <u>A</u> utomated <u>R</u> ule <u>M</u> utation <u>S</u> olution	45
4.3.1	System Overview	45
4.3.2	Rule Ingestion	47
4.3.3	Rule Mutation	49
4.3.4	Rule Optimization	51
4.3.5	Rule Adaptation	52
4.4	Implementation	54

4.4.1	LLM Models	54
4.4.2	Public Sigma Rules Repository	55
4.4.3	Knowledge Base	55
4.5	Evaluation	55
4.6	Discussion	59
5	Discussion: Toward a Proactive APT Detection Lifecycle	61
5.1	What AUTOSIGMA and ARMS each contribute	61
5.2	How the two systems connect	63
5.3	Unified detection engineering lifecycle	63
5.4	Limitations	64
6	Conclusion	66
6.1	Conclusion	66
6.2	Future Research Directions	68
	Appendix A LLM Prompting for AUTOSIGMA	69
	Appendix B Proof of Concept (PoC) (AUTOSIGMA)	71
	Appendix C CTI Reports Summary	73
	Appendix D Local LLMs Versus Cloud LLMs	75
	Appendix E Average Evaluation Cross Multiple APTs	77
	Appendix F Proof of Concept (PoC) (ARMS)	79
	Bibliography	81

List of Figures

Figure 1.1	Threat hunting process [1] with emphasis on our work focus (★).	4
Figure 3.1	Excerpt from Mandiant’s APT41 threat report [2]. This example illustrates the unstructured nature and narrative style of CTI reports, which pose challenges for automated rule extraction.	14
Figure 3.2	High-level overview of AUTOSigma and its components (components shown by blue are based on LLM).	17
Figure 3.3	Outputs of attack scenario enrichment and step decomposition.	22
Figure 3.4	Comparison of semantic similarities between pairs of inputs and pairs of generated Sigma rules across different APT41 reports.	38
Figure 3.5	Effect of temperature on semantic similarity.	39
Figure 3.6	Effect of model fine-tuning on hallucination.	39
Figure 4.1	Illustration of the problem statement and motivation behind ARMS.	42
Figure 4.2	High-level overview of ARMS and its components.	46
Figure 4.3	Illustrative example of a generated AST of a Sigma rule.	48
Figure 4.4	Example of three-dimensional variant discovery.	50
Figure 4.5	Example of the Infrastructure JSON file.	51
Figure 4.6	Overview of the prompt given to the RAG-based Adaptor Agent.	53
Figure 4.7	Impact of three-dimensional proximity space on the rule fidelity.	59
Figure 4.8	Impact of rule optimization on rule relevance.	59

Figure 5.1 Pipeline view of the unified workflow: AUTOSigma produces Sigma seed rules from narrative CTI, and ARMS expands these seeds into infrastructure-aware variant sets suitable for deployment.	63
Figure A.1 Example of prompts used by AUTOSigma.	69
Figure A.2 LLM-as-a-Judge prompts.	70
Figure B.1 Screenshot of the AUTOSigma PoC interface.	71
Figure F.1 Screenshot of the ARMS PoC interface.	79

List of Tables

Table 2.1	Comparison of representative CTI knowledge extraction solutions and their outputs, augmentation mechanisms, and extracted artifacts.	8
Table 2.2	Comparison with representative state-of-the-art systems.	10
Table 3.1	Average results of AUTOSigma against LLMCLOUDHUNTER using cloud-security blogs.	30
Table 3.2	Number of Sigma rules generated per APT41 reports.	33
Table 3.3	IoC-level coverage comparison between APT41 reports and generated Sigma rules.	34
Table 3.4	MITRE ATT&CK-level coverage comparison between APT41 reports and generated Sigma rules.	35
Table 3.5	BERTScore comparison between the input CTI report and the generated Sigma rules.	36
Table 3.6	Comparison of rule validity generated by different models across different APT41 reports.	37
Table 3.7	Comparison of rule validity by different models across different APT41 reports.	39
Table 4.1	Correctness of rules generated by ARMS compared with baselines.	56
Table 4.2	Correctness of rules generated by ARMS compared with baselines.	57
Table 5.1	How AUTOSigma and ARMS complement each other as stages in a unified detection engineering lifecycle.	62
Table C.1	Summary of CTI reports used in the evaluation.	74
Table E.1	Average results across APT groups for each evaluated model.	78

Chapter 1

Introduction

1.1 The Threat Landscape and Cyber Threat Intelligence

Advanced Persistent Threats (APTs) continue to pose a significant and evolving risk to organizations and critical infrastructure. Public reports from incident response and threat intelligence consistently describe a landscape in which capable adversaries pursue speed, stealth, and persistence, and where cyber incidents increasingly intersect with geopolitical tension and mature criminal ecosystems [3–7]. In response to this threat landscape, defenders need to gain understanding quickly, translate that understanding into detection content, and maintain coverage as adversaries change and adapt their techniques. In practice, defenders rely heavily on Cyber Threat Intelligence (CTI) reports (*e.g.*, APT reports). These reports document tactics, techniques, and procedures (TTPs), tooling, and infrastructure, and often capture high-value narrative context about how an attack occurred [3, 4]. However, CTI reports are typically written for human consumption. NIST explicitly characterizes a threat intelligence report as a prose document describing TTPs, actors, targeted systems, and other threat-relevant information [8, 9]. As a result, the operationalization of CTI often requires significant manual work.

1.2 Rule-based detection and the role of Sigma

The rapid use of detection rules in industry further highlights the need for adaptable and relevant detection logic. In practice, security teams often need to write new detection rules (*e.g.*, Sigma) manually to support proactive threat hunting and to address diverse operational contexts. Translating these unstructured threat descriptions into concrete detection content (*e.g.*, Sigma rules) is thus a bottleneck in a proactive defense stance when the Sigma rules can be generated from internal and external threat intelligence reports [10]. Existing CTI reports often emphasize known indicators of compromise (IoCs) which can be useful for detection but have limited longevity. To counter evolving threats, analysts must extract deeper patterns (*e.g.*, attack sequences or behaviors) from narrative text beyond transient indicators. Rule-based detection remains central to security operations and threat hunting. Organizations codify detection logic as SIEM queries, platform-specific rules, or telemetry-bound signatures, and then iteratively refine them as new threats emerge. Sigma has become a practical standard in this workflow by offering a vendor-agnostic, open detection rule format, supported by an ecosystem of tools and rule repositories [11, 12]. In particular, Sigma enables sharing detection intent in a portable form and then converted into platform-specific queries using available converters [11, 13]. Complementarily, the MITRE ATT&CK framework provides a widely used behavioral taxonomy for describing real-world adversary tactics and techniques, enabling threat-informed defense reasoning and coverage analysis [14]. Despite these advances, two bottlenecks limit how effectively CTI can be translated into resilient hunting capability.

1.3 The Dual Bottleneck in Detection

1.3.1 Bottleneck 1: The Manual Gap in CTI-to-rule Conversion

Converting unstructured threat narratives into actionable detection rules remains a bottleneck in maintaining a proactive defense posture. The traditional approach of manually creating Sigma rules is not only labor-intensive and prone to errors but also demands considerable domain expertise, which limits scalability and the ability to quickly respond to new threats. Although CTI often includes indicators of compromise, extracting more profound behavioral patterns from narrative

descriptions requires extensive manual work to develop effective defenses.

Prior efforts have tackled parts of this problem. For example, TTPDRILL [15] uses an ontology-based approach to parse CTI text, automatically extracting threat actions and mapping them to MITRE ATT&CK techniques. THREATRAPTOR [16] builds on this idea by linking extracted indicators and actions into a threat behavior graph that can be searched against system logs. Similarly, LADDER [17] aggregates attack patterns across multiple CTI sources to capture multi-stage intrusions. These systems demonstrate the value of structuring CTI, but they fall behind in producing ready-to-deploy detection rules. They often require well-structured input or significant manual curation, and they do not integrate with a rule repository to ground outputs in known detection logic. More recently, large language models (LLMs) have been applied to accelerate CTI to rule generation. Work in [18–21] review how LLMs and generative AI are transforming cybersecurity, including threat intelligence extraction and rule generation. These studies suggest that LLMs show promise in interpreting natural-language reports, but also highlight that automated rule synthesis remains immature. These approaches are promising, but detection engineering has constraints that make naive generation risky. In this domain, syntactic correctness is necessary but not sufficient, and semantic drift or unsupported assumptions can translate directly into unusable detections [22]. This motivates pipelines that explicitly ground generation in known rule structures and incorporate validation steps rather than relying on a single-pass model output.

1.3.2 Bottleneck 2: The Fragility of Static Rules Against Attack Variants

Even when a rule is correct, it is usually linked to a specific instance of an attack. Adversaries can achieve their objectives by altering the methods used, such as changing tools, execution paths, or platforms [4, 6, 7]. This fragility results in detection blind spots, and empirical studies indicate that commonly used rule sets can often be bypassed with minimal modifications, underscoring the operational risks of relying on static detections in dynamic environments [23]. Moreover, enterprises vary in their operating system configurations, logging coverage, schemas, and legitimate administrative activities. Threat hunting practices reveal ongoing challenges in developing and maintaining automation that remains aligned with the specific environment in which it is deployed [24]. Without awareness of the infrastructure, external or generic rules may become redundant,

irrelevant, or excessively noisy.

Previous research has investigated methods to reduce the cost of generating initial detections, including approaches that apply Natural Language Processing (NLP) and Large Language Models (LLMs) to generate detection rule candidates from unstructured sources [20, 21]. Other attempts tend to address the operational cost of large rule sets [25] by optimizing and consolidating rule sets to reduce redundancy and improve efficiency. These approaches do not address a fundamental challenge frequently encountered in detection operations: (i) maintaining coverage when attackers alter their techniques, (ii) ensuring that the rules remain valid and compatible with the target environment. Although evasion-aware rule hardening [26] can enhance robustness against bypass patterns, (iii) it does not address technique substitution and behavioral drift outside the original rule’s scope.

Together, these challenges highlight the need for automation not only in converting CTI into detection rules, but also in evolving and mutating those rules to remain effective as attackers change techniques in different environments.

1.4 Contributions

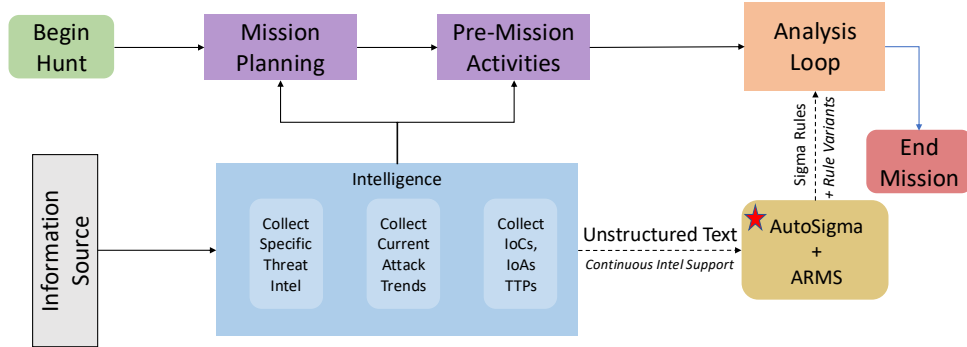


Figure 1.1: Threat hunting process [1] with emphasis on our work focus (★).

This thesis conceptualizes APT hunting as a detection engineering lifecycle that begins with transforming CTI into deployable detections and then adapts these detections to remain effective despite technique changes and environmental constraints. Lifecycle refers to the process of generating detection rules from threat intelligence and continuously adapting those rules as threats and environments evolve. We implement this lifecycle through two systems: AUTO SIGMA and ARMS, each

offering distinct contributions discussed in the relevant chapter. Fig. 1.1 shows how AUTOSigma and ARMS assist and enhance the threat hunting process [1]. The primary contributions of this thesis are:

- AUTOSigma, an automated solution that converts narrative CTI into deployable Sigma rules by integrating contextual analysis, attack comprehension, template-based grounding over validated rule repositories, and iterative rule refinement with explicit validation.
- ARMS, an automated rule mutation solution that creates infrastructure-aware detection variants to maintain coverage when adversaries alter techniques, while ensuring compatibility with the telemetry and constraints of the target environment.
- A comprehensive view of CTI-driven, rule-based APT hunting as a lifecycle, connecting CTI operationalization and rule evolution, and clarifying where automation is essential and where control and validation are crucial.

1.5 Thesis organization

The remainder of this thesis is organized as follows. Chapter 2 reviews the background of threat hunting and CTI, the foundations of Sigma and ATT&CK, and relevant research. Chapter 3 details AUTOSigma, including its design, architecture, implementation, and evaluation. Chapter 4 discusses ARMS, covering the architecture, methodology, implementation, and evaluations. Chapter 5 explores how AUTOSigma and ARMS integrate as a cohesive workflow for proactive APT hunting. Chapter 6 concludes with limitations and directions for future research.

Chapter 2

Background and Related Work

2.1 Threat hunting and detection engineering context

Threat hunting as an operational practice: Threat hunting is a proactive workflow in which analysts form hypotheses, interrogate telemetry, and iterate based on findings to identify threats that escape purely reactive monitoring. Interview evidence suggests threat hunting is operationally organized as planning, collection, analysis, and refinement under resource constraints [24]. In this context, the time required to operationalize new intelligence and the challenge of maintaining detection tend to be a recurring operational concern.

Detection engineering and rule-based defenses: Detection engineering involves the creation, implementation, validation, and upkeep of detection logic across monitoring systems. Organizations typically employ detection logic as SIEM queries, endpoint rules, and telemetry-bound signatures, continuously updating these elements to manage false positives and maintain coverage as environments evolve [27, 28]. The deployment of detection logic across diverse platforms results in high maintenance costs and limited reuse, prompting research that views detection rules as primary artifacts and aims to automate parts of their lifecycle.

2.2 Cyber Threat Intelligence and CTI extraction tools

CTI and the operationalization gap: CTI is a primary channel for sharing what defenders learn about threat actors and their TTPs [29]. CTI is frequently disseminated as narrative reports written for human analysts, which introduces a structural gap between intelligence and deployable detection logic. Surveys on CTI sharing and management highlight persistent challenges in scaling CTI use across organizations, including standardization and operational integration [10, 29].

CTI knowledge extraction: Early methods for knowledge extraction from CTI include TTP-DRILL [15], which uses unsupervised NLP to extract threat actions from reports and map them to MITRE ATT&CK techniques. EXTRACTOR [30] advances this by using BERT-BiLSTM to convert unstructured CTI into formal evidence graphs for threat hunting. THREATRAPTOR [16] combines action and indicator extraction into structured behavior graphs. CASIE [31] and OPEN-CYKG [32] use BiLSTM-based models to extract entities and relations from reports to populate threat knowledge graphs. Later systems such as SECIE [33], TriCTI [34], and CYBERENTREL [35] extend this direction with model and pipeline improvements for cybersecurity extraction tasks. Overall, these systems improve extraction quality and representation, but their outputs are typically graphs or taxonomies rather than deployable detection rules. As a result, practitioners must still manually translate extracted knowledge into operational detection logic.

LLM-based extraction and threat reasoning: More recent work applies large language models to CTI extraction and reasoning tasks. [36–38] explore LLM-driven extraction and structuring from informal or mixed sources, while LADDER [17] emphasizes multi-stage attack pattern extraction. SHIELD [39] demonstrates APT detection and explanation with iterative reasoning, and related approaches explore end-to-end extraction pipelines [40]. These systems strengthen the ability to derive structured knowledge from narrative sources, but they typically output graphs, entities, or taxonomies rather than deployable detection rules.

Table 2.1 summarizes representative CTI extraction systems and highlights recurring limitations from the perspective of operational detection engineering.

Table 2.1: Comparison of representative CTI knowledge extraction solutions and their outputs, augmentation mechanisms, and extracted artifacts.

Reference	Year	Technique	Dataset	Target Env.	Output Type	Output			Augmentation		Extraction		
						Act.	Rob.	WF	KBs	Temp.	Rel.	IoCs	TTPs
NLP-based Solutions													
TTPDRILL [15]	2017	Unsupervised NLP	Symantec	On-prem	STIX	✗	✗	✗	✗	✗	✓	✓	✓
THREATRAPTOR [16]	2021	Unsupervised NLP	DARPA TC	On-prem	Behavior Graph + SQL	✗	✗	✗	✗	✗	✓	✓	✓
BERT-based and BiLSTM Solutions													
CASIE [31]	2020	BiLSTM	CyberWire	On-prem	Knowledge Graph	✗	✗	✗	✗	✗	✓	✓	✗
EXTRACTOR [30]	2021	BERT-BiLSTM	Mixed Sources	On-prem	Behavior Graph	✗	✗	✗	✗	✗	✓	✓	✓
OPEN-CYKG [32]	2021	BiLSTM	MalwareDB	On-prem	Knowledge Graph	✗	✗	✗	✗	✗	✓	✗	✓
SecIE [33]	2022	BERT	CVE	On-prem	Knowledge Graph	✗	✗	✗	✓	✗	✓	✓	✗
TriCTI [34]	2022	BERT	Custom	On-prem	Labeled IoCs	✗	✗	✗	✗	✗	✗	✓	✗
LADDER [17]	2023	BERT	Custom	On-prem	Knowledge Graph	✗	✗	✗	✓	✗	✓	✓	✓
CYBERENTREL [35]	2024	RoBERTa-BiGRU-CRF	Custom	On-prem	Knowledge Graph	✗	✗	✗	✓	✗	✓	✓	✗
FLOWGUARDIAN [40]	2025	BERT	Custom	On-prem	TestFlow	✗	✗	✗	✗	✗	✓	✓	✓
LLM-based Solutions													
PURBA, CHU [36]	2023	GPT-3.5	Twitter Posts	On-prem	Labeled IoCs	✗	✗	✗	✗	✗	✗	✓	✗
acTion [38]	2023	GPT-3.5	Custom	On-prem	STIX	✗	✗	✗	✓	✗	✓	✓	✓
Liu et al. [37]	2023	ChatGPT	Custom	On-prem	Knowledge Graph	✗	✗	✗	✓	✗	✓	✓	✓
LLM-TIKG [41]	2023	LLaMA-2-7B (fine-tuned)	Custom	On-prem	Knowledge Graph	✗	✗	✗	✓	✗	✓	✓	✓
Fengrui et al. [42]	2024	LLaMA-2-7B (fine-tuned)	ATT&CK STIX Data	Hybrid	MITRE ATT&CK TTPs	✗	✗	✗	✓	✗	✓	✓	✓
INTELEX [21]	2024	LLM	APT CTI	On-prem	Knowledge Graph + Sigma Rule	✓	✗	✗	✓	✗	✓	✓	✓
LLMCLOUDHUNTER [20]	2025	GPT-4o	Custom	Cloud	Sigma Rules	✓	✗	✗	✗	✗	✓	✓	✓
AUTOSigma	2025	LLM (LLM-as-a-Judge)	Custom	Cloud & On-prem	Flow of Sigma Rules	✓	✓	✓	✓	✓	✓	✓	✓

Act.: Actionable

KBs: External Knowledge Base

Rob.: Robust to input quality

Temp.: Template retrieval

WF: Workflow support

Rel.: Relations

2.3 MITRE ATT&CK and behavioral indexing

ATT&CK as a shared behavioral vocabulary: MITRE ATT&CK provides a widely used knowledge base for describing adversary behavior through tactics and techniques [43]. It provides a shared vocabulary that helps normalize CTI narratives and detection intent across sources. This helps defenders reason about coverage, technique overlap, and detection gaps.

Mapping CTI, vulnerabilities, and artifacts to ATT&CK: A common research pattern is to map extracted CTI artifacts to ATT&CK techniques. TTPDRILL [15] performs technique mapping as part of extraction, and later work explores automated discovery and mapping of tactics and techniques from CTI [44]. Related research bridges vulnerability information and ATT&CK modeling, including semantic mapping of CVEs to ATT&CK for cybersecurity applications [45, 46]. Few-shot and transfer learning approaches have also been explored for technique or tactic identification in text [42, 47].

Behavioral indexing improves interoperability, but mappings can be ambiguous when a narrative

lacks observable details, or when multiple techniques plausibly explain a described action. Additionally, ATT&CK describes adversary behavior and does not prescribe telemetry semantics or rule construction.

2.4 Sigma and detection rule portability

Portability across heterogeneous detection stacks: Detection logic is implemented across diverse systems, and differences in schemas and query languages complicate reuse and deployment. Sigma addresses part of this challenge by defining a platform-agnostic rule representation for sharing and conversion [48]. Operational work on SIEM systems reflects broad variability in usage patterns and telemetry integration, which reinforces the need for portable rule representations [27]. Practical detection engineering references also emphasize that detection content must be validated against data sources and operational constraints, not only authored correctly [28].

Sigma ecosystem and conversion: Sigma is supported by community tooling that facilitates rule management and conversion [49]. In practice, Sigma rules often serve as shared detection intent, while conversion tools translate that intent into backend-specific implementations. This portability makes Sigma a natural intermediate artifact for CTI-to-rule synthesis and downstream rule adaptation.

2.5 Automated rule generation

LLM-based rule generation: Recent work has started to bridge the gap between unstructured sources and detection by generating initial detection rules. LLMCLOUDHUNTER [20] uses large language models to generate Sigma rule candidates from cloud-focused open-source intelligence and validates backend conversion success. INTELEX [21] extracts attack-level intelligence from CTI reports and demonstrates downstream Sigma rule generation as part of its evaluation. Practitioner-oriented systems such as SIGMAGEN [50] similarly aim to reduce the time for rule generation through automated extraction and ATT&CK mapping. While these efforts reduce the cost of producing detections, they largely treat generation as a one-pass task and produce rules that are tightly tied to the described instance. They do not directly address proactive technique substitution, cross-platform

Table 2.2: Comparison with representative state-of-the-art systems.

Category	Reference	Variant Discovery	Actionable Logic	Infrastructure Awareness
Static Rule Generation	[20]	✗	✓	✗
	[21]	✗	✓	✗
	[53]	✗	✓	✗
	[54]	✗	✓	✗
	[55]	✗	✓	✗
Rule Optimization	[25]	✗	✓	✓
Rule Mutation	ARMS	✓	✓	✓

behavioral drift, or feasibility under enterprise-specific telemetry constraints, which are necessary to keep static rules effective over time.

Knowledge-augmented generation and retrieval: A recurrent theme in LLM-based systems is the need to couple generation with retrieval and external knowledge so that outputs remain consistent with known patterns and constraints. Retrieval-Augmented Generation (RAG) provides one mechanism for grounding LLM outputs in retrieved references [51]. In cybersecurity contexts, knowledge-assisted RAG has also been explored [52]. These approaches are relevant because detection rule generation benefits from access to prior rules, schemas, and structured threat knowledge.

2.6 Rule robustness, mutation, and optimization

Evasion and robustness of static rules: Rule-based detections can be bypassed when adversaries exploit weaknesses in matching conditions or when defenders encode brittle assumptions. Work on evasion against detection rules demonstrates that even expert-written rules can be evaded [26], motivating systematic approaches to strengthen rule robustness. While evasion-aware hardening improves resilience against bypass patterns, it does not necessarily address technique substitution where an adversary changes execution primitives or shifts platforms.

Rule-set optimization and operational burden: Beyond initial rule creation, recent work has highlighted the operational burden of maintaining large rule sets. RULEGENIE [25] focuses on SIEM

rule set optimization by identifying similar or redundant rules and recommending consolidation and refinement to reduce alert fatigue. This setting is closely related to the rule lifecycle perspective adopted in this thesis in that it treats detection rules as the primary artifact and targets real-world operational constraints. However, RULEGENIE optimizes existing rule sets rather than generating behaviorally equivalent rule variants under technique substitution, and it does not explicitly filter outputs according to an enterprise infrastructure profile. Table 2.2 summarizes representative systems across rule generation, optimization, and mutation research directions, highlighting the limited support for infrastructure-aware variant discovery in existing approaches.

2.7 Research gaps and motivations

Gap 1: from structured CTI to deployable detection logic: CTI extraction systems have made substantial progress in extracting entities, relations, and behavior graphs [15–17, 30, 31]. However, many approaches stop at structured representations and do not consistently produce deployable detection logic as an output artifact. This gap motivates CTI-to-rule synthesis pipelines that treat Sigma rules as first-class outputs [48].

Gap 2: static rule generation and instance dependence: Initial rule generation efforts reduce authoring cost, but they often produce detections tied to the described instance and do not explicitly address long-term coverage under technique substitution or cross-platform drift [20, 21]. This motivates generation workflows in which grounding and validation are integrated to reduce semantic drift and improve deployability [56, 57].

Gap 3: robustness beyond bypass hardening: Evasion-aware hardening highlights that detection logic can be fragile and should be validated against bypass strategies [26]. However, hardening does not fully address behavioral drift that falls outside the scope of the original rule when adversaries switch tools or techniques. This motivates rule mutation approaches that treat technique substitution as a core requirement [58, 59].

Gap 4: infrastructure-aware adaptation and lifecycle integration: Finally, operational studies emphasize that detection engineering depends on telemetry availability and organizational processes [1, 24]. Existing work does not provide an integrated lifecycle that begins at narrative CTI, yields

grounded Sigma detections, and then adapts those detections into environment-compatible variants under technique substitution . These gaps motivate the two-stage thesis workflow: `AUTOSigma` for CTI-to-Sigma synthesis and `ARMS` for infrastructure-aware rule mutation and adaptation.

Chapter 3

AUTO SIGMA: Automated Sigma Rule Generation

3.1 Motivation and Problem Statement

3.1.1 Motivation

Sigma rules are a widely adopted, vendor-agnostic signature format (written in YAML) for encoding CTI as detection logic across diverse SIEM platforms [10]. This portability makes Sigma particularly valuable, but also makes relevant rule creation even more critical. In practice, security teams often need to write new Sigma rules to support proactive threat hunting and to address diverse operational contexts. Traditional methods for manually crafting Sigma rules are not only time-consuming and error-prone, but also require significant domain expertise, limiting their scalability and responsiveness to emerging threats. This manual workflow cannot scale and urgently calls for automation [29]. Despite the existence of open-source Sigma rule repositories, these repositories cannot keep pace with the rapid evolution of adversarial tactics.

CTI reports (*e.g.*, Fig. 3.1) are inherently free-form and heterogeneous, often blending detailed attack steps with benign observations and inconsistent formatting. Automated parsing of such reports into Sigma’s structured conditions is extremely challenging. Prior research has proposed various NLP-driven extraction techniques, but these generally yield only limited structured outputs

(such as identification of IoCs, or MITRE ATT&CK techniques) that still require extensive manual interpretation [15,30]. Moreover, even the existing Sigma rule corpus is far from complete: study [20] found that only 60 of 193 MITRE ATT&CK patterns had a corresponding Sigma rule (approximately 31% coverage). In other words, many behaviors documented in CTI may have no ready-made detection rule at all, forcing analysts to fill in gaps manually.

Starting on January 20, 2020, APT41 used the IP address 66.42.98[.]220 to attempt exploits of Citrix Application Delivery Controller (ADC) and Citrix Gateway devices with CVE-2019-19781 (published December 17, 2019). The initial CVE-2019-19781 exploitation activity on January 20 and January 21, 2020, involved execution of the command 'file /bin/pwd', which may have achieved two objectives for APT41. First, it would confirm whether the system was vulnerable and the mitigation wasn't applied. Second, it may return architecture-related information that would be required knowledge for APT41 to successfully deploy a backdoor in a follow-up step.

Figure 3.1: Excerpt from Mandiant’s APT41 threat report [2]. This example illustrates the unstructured nature and narrative style of CTI reports, which pose challenges for automated rule extraction.

3.1.2 Problem Statement

Most methods [20, 21, 38] rely heavily on the quality and structure of the input report, which means they produce inconsistent results when faced with unstructured or poorly formatted documents. Other approaches [17, 41] lack contextual enrichment; they do not incorporate additional knowledge sources to augment the information in the report, therefore limiting the accuracy and relevance of the generated rules. Moreover, the insights extracted by these systems often lack the granularity and structured format necessary for direct integration into SIEM environments. Consequently, security teams must manually refine or complete these partially generated rules, undermining the goal of full automation. More fundamentally, there is a conspicuous gap in research when it comes to translating narrative threat intelligence into actionable content. Only a few studies [20, 21] have attempted to bridge this gap, and they face several obstacles: (i) the inherent complexity of converting natural-language descriptions of attacks into the formal structure of Sigma rules, (ii) the difficulty of correlating the sequence of extracted attack steps with existing detection rule patterns, which is essential to ensure that any generated rule is consistent with known attack behaviors and truly relevant. These challenges have so far prevented the development of a solution capable of producing high-quality, ready-to-use Sigma rules from unstructured threat reports.

3.1.3 Contribution and Advantages of AutoSigma

Contribution: CTI reports, attack trends, and behavioral indicators drive analysis and mission planning. AutoSigma automates a core bottleneck in this loop by transforming narrative threat intelligence into structured Sigma rules. Although recent work has made progress toward automating certain CTI processing tasks, there is still no end-to-end solution capable of taking raw narrative reports and producing validated Sigma rules, aligned with MITRE ATT&CK and applicable to different attack types, whether living-off-the-land, cloud-based, or otherwise. To address this gap, we present AutoSigma, an end-to-end solution that transforms unstructured CTI reports into precise, deployable Sigma rules. Our contributions can be summarized as follows:

- *Fully automated pipeline:* Unlike prior work [20, 21], AutoSigma performs contextual analysis, template-driven rule retrieval, and generative refinement in a unified and automated pipeline that helps in reducing manual effort.
- *Attack scenario enrichment:* AutoSigma enhances attack understanding by using external cybersecurity knowledge bases to fill in missing details and extend raw threat descriptions, which enables robust rule generation even with sparse input.
- *Attack decomposition:* AutoSigma decomposes complex attack scenarios into smaller, atomic steps aligned with known detection logic. This improves the accuracy, granularity, and modularity of the generated Sigma rules.
- *Template-based rule grounding:* By retrieving similar Sigma rules from validated repositories, AutoSigma uses them as templates to guide and constrain rule generation, which ensures syntactic correctness and contextual alignment with industry best practices.
- *LLM-as-a-Judge refinement:* AutoSigma incorporates a dual-LLM feedback mechanism that iteratively validates and improves candidate rules, which helps in mitigating hallucination, enhancing quality, and ensuring rule validity.
- *Real-world validation:* We empirically evaluate AutoSigma on multiple public security blogs and also APT reports (APT41, APT28, and APT29) against state-of-the-art solution and LLM models,

demonstrating high-fidelity rule generation, accurate threat behavior capture, and readiness for deployment in operational SIEM environments.

Advantages of AUTOSigma: With its LLM-driven refinement mechanism, AUTOSigma offers several advantages over prior approaches. First, unlike LLMCloudHunter [20] that requires highly detailed reports, AUTOSigma can compensate for incomplete intelligence by enriching attack tests with external knowledge bases, and thus accelerate rule generation and provide more threat coverage. Second, unlike traditional single-pass generation or standard self-refinement methods [20], AUTOSigma introduces a feedback refinement loop driven by a dual LLM-as-a-Judge that aims to generate rules and refine them for better accuracy. This approach ensures consistent formatting and syntax of Sigma rules, which reduces human errors and improves detection accuracy. Finally, AUTOSigma eases the operationalization of threat intelligence to be used in security operations without requiring expert rule engineers for every new threat report.

3.2 Threat Model and Assumptions

In-scope threats: Our solution targets sophisticated, multi-stage cyberattacks, such as APTs, that are described in unstructured documents (*e.g.*, threat reports, incident response documentation). These attacks typically exploit software vulnerabilities (*e.g.*, Common Vulnerabilities and Exposures – CVE), leverage known adversary tactics and techniques, and proceed through a kill-chain involving initial access, lateral movement, and impact.

Out-of-scope threats: AUTOSigma does not aim to detect physical security incidents, insider threats not documented in reports, or attacks that do not leave observable artifacts suitable for Sigma rule generation. Similarly, zero-day threats with no previous contextual documentation are not within the immediate scope of detection unless indirectly referenced in CTI.

System Assumptions: We assume that the input is an unstructured threat report that describes one or more attack scenarios. Although our experiments focus on APT reports, our work is not limited to them; the input can be any textual description of malicious activity, including incident response reports, blog posts, or informal threat notes. Our methodology does not require the input text to be cleanly structured or annotated. We assume input reports are authentic and not intentionally

manipulated. We assume that each report may contain multiple discrete attack steps. As such, the system is expected to generate not one, but a sequence of Sigma rules, each corresponding to a specific attack action or tactic. These rules may vary in granularity depending on the level of detail available in the input. Finally, we assume the correctness of the external knowledge bases that the solution is leveraging.

3.3 Automated Sigma Rules Generation

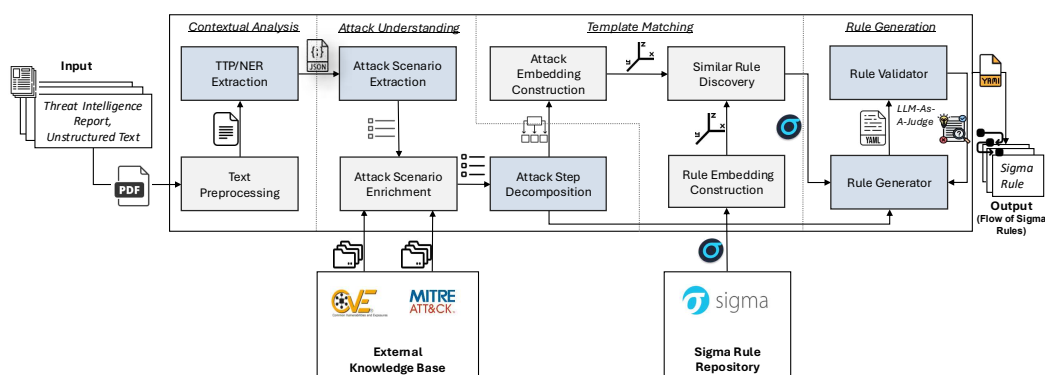


Figure 3.2: High-level overview of AUTOSigma and its components (components shown by blue are based on LLM).

3.3.1 System Overview

Overview: AUTOSigma, depicted in Fig. 3.2, is designed to automate the transformation of APT reports into high-quality, actionable Sigma rules. The architecture follows a modular pipeline approach, consisting of four main components. The first component of AUTOSigma is Contextual Analysis (Section 3.3.2), where it performs text preprocessing and applies Named Entity Recognition (NER) and TTP extraction to identify key elements within the unstructured threat report. Next, in the Attack Understanding component (Section 3.3.3), the extracted entities and tactics are analyzed to infer possible attack scenarios. These scenarios are then enriched using external knowledge bases such as MITRE ATT&CK and NVD, allowing AUTOSigma to incorporate intelligence beyond what is explicitly stated in the report—thereby mitigating the limitations imposed by poor-quality or incomplete input. Subsequently, complex attacks are decomposed into atomic attack steps, enabling

precise reconstruction of the attack chain and ensuring that generated rules are both comprehensive and granular. Following this, the Template Matching component (Section 3.3.4) employs a semantic similarity search over a curated repository of validated Sigma rules. By embedding both the extracted attack step and the rule corpus into a shared vector space, AUTOSigma retrieves the closest matching rule template, grounding its output in previously vetted detection logic. This retrieval-based grounding is a novel contribution that ensures generated rules inherit the structural correctness and semantic intent of battle-tested Sigma patterns—something not addressed in prior approaches. Finally, in the Rule Generation component (Section 3.3.5), AUTOSigma uses an LLM-as-a-Judge paradigm to iteratively refine and validate each Sigma rule. A generation model proposes a candidate rule, while a validator model critiques and suggests corrections. This adversarial loop is repeated until the output satisfies both syntactic validity and semantic alignment with the source scenario. Through this pipeline, AUTOSigma combines multi-stage contextual reasoning, external enrichment, template-driven rule retrieval and iterative validation to produce robust Sigma rules that are both immediately deployable and semantically faithful to the underlying threat report.

Novelty: AUTOSigma introduces multiple novel elements that distinguish it from prior approaches. First, it performs end-to-end automation of the Sigma rule generation process, starting from raw unstructured threat reports to producing relevant, deployable rules. Second, it integrates external knowledge sources such as the MITRE ATT&CK knowledge base and the National Vulnerability Database (NVD)¹ to enrich attack context and fill in gaps from incomplete reports. Third, it constructs the entire attack chain by decomposing each scenario into a sequence of attack steps, each mapped to a Sigma rule. Fourth, it leverages an up-to-date repository of official Sigma rules to retrieve and adapt existing templates through similarity search. Finally, a dual-LLM feedback loop, where a rule generator and a rule validator iteratively collaborate to refine the quality of the generated sigma rules.

Design Challenges: Designing AUTOSigma required addressing several challenges. CTI reports are typically written in a free-form and lack consistent structure, making it difficult to directly extract entities or behavior sequences. The availability of high-quality, labeled cybersecurity data is limited, which constrains the use of traditional supervised models for tasks like NER. Additionally,

¹NVD: <https://nvd.nist.gov/>

large language models, while powerful, are prone to hallucinations or producing outputs that are syntactically incorrect or semantically misaligned with the input. Controlling these errors and handling failure cases gracefully was essential to ensuring pipeline reliability. AUTOSigma addresses these challenges by contextual analysis and leveraging external knowledge bases, uses LLMs for tasks like NER extraction and also uses LLM-as-a-Judge method so that it will be able to control the effect of hallucinations.

3.3.2 Contextual Analysis

The first component of AUTOSigma is responsible for extracting meaningful threat intelligence from raw threat reports. Since these reports vary widely in structure and format, this module applies preprocessing techniques to ensure consistency, remove irrelevant content, and isolate the information most pertinent to subsequent analysis.

Text Preprocessing:

AUTOSigma ingests threat reports and performs rigorous text preprocessing, which prepares the data for further analysis. This phase involves several NLP techniques [60] to refine raw, unstructured data: (a) extracting the text given a threat report; (b) text cleaning, which removes the irrelevant data from the text and prepares the text for further analysis; (c) normalization, which standardizes the data for consistency and improved algorithm performance; and (d) segmentation, which allows us to feed the language models with input in manageable pieces, optimizing the performance of the LLMs in our architecture by providing context segment-by-segment instead of in one large block [61].

TTP and NER Extraction:

Various NLP solutions have been proposed for extracting named entities and mapping tactics, techniques in cybersecurity contexts, such as PELAT [47], SMET [45], and Adema [44]. However, these approaches often rely on annotated datasets and may lack adaptability to diverse, unstructured threat reports. To address these limitations, AUTOSigma uses LLM for NER extraction, leveraging its capabilities to generalize from limited data and understand complex contexts. We also experimented with a BERT-based NER model [62], but the LLM approach yielded better results given the limited

training data available for these entity types. Once the key entities are extracted from the report, AUTOSigma analyzes them to find tactics, techniques, and aligns the information with the MITRE ATT&CK knowledge base. This alignment provides a precise categorization of the described attacker behaviors, ensuring that subsequent processing steps have a structured understanding of the threat context.

Definition 1. Let $\mathcal{D}$ denote the set of unstructured CTI documents (threat reports). Let $\mathcal{S}$ be the set of preprocessed (normalized, cleaned, and segmented) texts and $\mathcal{X}$ be the set of contextual information (NER, IoCs, MITRE ATT&CK techniques/tactics) that AUTOSigma extracted. The contextual representation space is $\mathcal{C} \subseteq 2^{\mathcal{S}} \times 2^{\mathcal{X}}$. The contextual analysis component is modeled as a function $F_{\text{ctx}} : \mathcal{D} \rightarrow \mathcal{C}$, which, for every report $d \in \mathcal{D}$, produces a contextual representation $f_{\text{ctx}}(d) = c_d = (S_d, X_d)$, where $S_d \subseteq \mathcal{S}$ is the set preprocessed text and $X_d \subseteq \mathcal{X}$ is the set of extracted contextual information for d .

Example 1. Consider the following excerpt from a threat report: *"APT41 leveraged CVE-2019-19781 to deploy Cobalt Strike beacons, facilitating lateral movement across the network."* In this case, AUTOSigma identifies APT41 as the threat actor, CVE-2019-19781 as the exploited vulnerability, and Cobalt Strike as the malware used in the operation. Based on this, it infers the corresponding MITRE ATT&CK techniques: Exploit Public-Facing Application (T1190), Command and Control through Beacons (T1071.001), and Lateral Movement via Remote Services (T1021).

3.3.3 Attack Understanding

While NER extraction and MITRE ATT&CK technique mapping provide a foundational understanding of threat reports, they are often insufficient for generating high-quality, specific Sigma rules. This is because the raw extracted entities alone may lack context, depth, or completeness—especially in cases where the report is vague or incomplete. Therefore, AUTOSigma performs an additional layer of analysis to construct a more coherent and actionable representation of the attack. In this phase, AUTOSigma goes beyond simple extraction—meaning it not only identifies entities and techniques, but also enriches them using external cybersecurity sources. For example, if a CVE is mentioned,

AUTOSigma queries the National Vulnerability Database (NVD)² to retrieve its detailed description, severity, and affected software. Similarly, if a known threat actor is referenced, information from the MITRE ATT&CK knowledge base is used to retrieve related tactics, tools, and historical campaigns. Furthermore, to ensure that the Sigma rules generated are both granular and actionable, this component decomposes complex attack descriptions into a sequence of atomic attack tests. Each test corresponds to a specific malicious behavior or tactic, and ultimately maps to a separate Sigma rule. This decomposition is critical for generating accurate and modular detection logic.

Attack Scenario Extraction:

Using the identified NER entities and TTPs, the system reconstructs meaningful attack scenarios mentioned in the report. This step synthesizes fragmented information into structured attack scenarios. For example, APT41 exploited CVE-2019-19781 to gain initial access to the target system by utilizing FTP to download a malicious payload.

These attack scenarios provide structured context for further enrichment and Sigma rule generation. By the end of this step, AUTOSigma produces a structured representation of the extracted intelligence, making it suitable for subsequent enrichment and detection rule generation.

Attack Enrichment:

To provide additional context to attack scenarios, AUTOSigma queries multiple cybersecurity knowledge bases. The enrichment process involves, but not limited to:

- *CVE Enrichment:* If an attack uses a CVE (*e.g.*, CVE-2019-19781), AUTOSigma retrieves relevant details such as the description of the vulnerability, CVSS score and severity rating, affected software, and versions. This information is sourced from external knowledge bases. In this work, we used the CVE repository as our knowledge base. The repository contains over 277,000 CVE records, each providing standardized information about publicly known cybersecurity vulnerabilities, including descriptions, severity scores, and affected products [46].

²NVD: <https://nvd.nist.gov/>

- *Threat Actor Enrichment:* We also use MITRE ATT&CK as a knowledge base, where AutoSIGMA queries it to extract additional information related to the attack. This information can be captured in different components, such as tools, malware, and software that a threat actor might use.

```

1 {
2   "attack_id": "Attack_1",
3   "description": "APT41 exploited CVE-2019-19781 to gain initial
4     access to the target system by utilizing FTP to download a
5     malicious payload.",
6   "related_tactics": ["Initial Access"],
7   "related_techniques": ["Exploitation of Remote Services"],
8   "indicators_of_compromise": ["CVE-2019-19781", "66.42.98[.]220"],
9   "cve_enrichment": [
10     {
11       "cve_details": {
12         "cve_id": "CVE-2019-19781",
13         "description": "An issue was discovered in Citrix
14           Application Delivery Controller (ADC) and Gateway 10.
15           5, 11.1, 12.0, 12.1, and 13.0. They allow Directory
16           Traversal.",
17         "cvss_score": 7.5,
18         "severity": "High",
19         "published_date": "2019-12-27",
20         "last_modified": "2025-02-04"
21       }
22     }
23   ],
24   "threat_actor_enrichment": {
25     "apt_group": "APT41",
26     "mitre_url": "https://attack.mitre.org/groups/G0092/",
27     "status": "Success"
28   }
29 }

```

(a) An example of attack scenario enrichment output.

```

1 {
2   "attack_scenarios": [
3     {
4       "attack_id": "Attack_1_1",
5       "description": "APT41 identifies a vulnerability in Citrix
6         Application Delivery Controller (ADC) and Gateway that
7         allows directory traversal.",
8       "related_tactics": ["Initial Access"],
9       "related_techniques": ["Exploitation of Public-Facing
10         Application"],
11       "indicators_of_compromise": ["Vulnerability CVE-XXXX-XXXX"]
12     },
13     {
14       "attack_id": "Attack_1_2",
15       "description": "APT41 exploits the directory traversal
16         vulnerability to access restricted directories.",
17       "related_tactics": ["Initial Access"],
18       "related_techniques": ["Exploitation of Vulnerability"],
19       "indicators_of_compromise": ["Unusual HTTP requests to
20         sensitive paths"]
21     },
22     {
23       "attack_id": "Attack_1_3",
24       "description": "APT41 uses FTP to download a malicious
25         payload onto the compromised system.",
26       "related_tactics": ["Execution"],
27       "related_techniques": ["Command and Scripting Interpreter"],
28       "indicators_of_compromise": ["FTP logs showing unusual file
29         downloads", "Malicious file hashes"]
30     }
31   ]
32 }

```

(b) An example of attack step decomposition output.

Figure 3.3: Outputs of attack scenario enrichment and step decomposition.

Attack Step Decomposition:

Given that an attack scenario often contains multiple attack behaviors or subgoals, it is not feasible to generate a single Sigma rule that effectively captures all of them. Therefore, AutoSIGMA decomposes each enriched scenario into smaller, discrete attack steps. This decomposition plays a critical role in aligning each step with existing detection logic more effectively. Unlike prior approaches [20, 21] that attempt to encapsulate an entire attack scenario within a single rule, our method breaks down complex threats into distinct, manageable components—enabling the generation of multiple, focused flows of sigma rules. This not only enhances the precision of each rule but also promotes modularity and scalability in detection coverage. Each step isolates a specific tactic or action taken by the adversary, which in the next phase will be mapped to a detection rule. To perform this decomposition, AutoSIGMA uses a prompting-based approach with an LLM, where the enriched attack description

is input and the model is guided to split the scenario into atomic substeps. This step ensures that every component of an attack is isolated and can be individually analyzed for detection logic, thereby eliminating ambiguity and improving the fidelity of the final detection rules. An example of the output for this step is illustrated in Fig. 3.3. In Fig. 3.3a, we observe the output of the enrichment phase, which includes extracted entities such as the CVE (lines 9-17), threat actor (lines 19-24), and contextual description. In Fig. 3.3b, the enriched scenario is decomposed into three smaller attack steps: the first involves the identification of a vulnerability in Citrix ADC (Initial Access), the second describes the actual exploitation attempt using that vulnerability, and the third captures the follow-up action of downloading a malicious payload using FTP (Execution phase). Each of these steps is now ready to be handled separately by the rule generation module.

Definition 2. Let $\mathcal{A}$ be the set of enriched and decomposed attack steps, where each $a \in \mathcal{A}$ describes a single malicious behavior together with its associated indicators, MITRE ATT&CK annotations, and enriched information (*e.g.*, CVE metadata). The attack understanding component is defined as $f_{\text{au}} : \mathcal{C} \rightarrow 2^{\mathcal{A}}$, mapping each contextual representation $c_d \in \mathcal{C}$ to a finite set of enriched attack steps $f_{\text{au}}(c_d) = A_d \subseteq \mathcal{A}$.

3.3.4 Template Matching

To improve the relevance and quality of the generated rules, we introduce a novel step: retrieving similar existing Sigma rules to be used as templates for generating new ones. In the third stage, AUTOSIGMA prepares to leverage existing knowledge of known detection rules by embedding both the attack steps derived from the report and a large collection of known and valid Sigma rules into a common vector space. By representing attack descriptions extracted from the attack understanding phase, and Sigma rules as numerical embeddings, AUTOSIGMA can perform efficient similarity searches to identify which known rules are most relevant to the attack steps at hand. This component bridges the gap between the enriched attack intelligence and the rule generation phase by providing a starting template for each detection rule AUTOSIGMA needs to generate.

Attack Embedding Construction:

Each attack description produced in the previous stage is converted into a vector embedding using a BERT-based model [62]. This transformation captures the semantic meaning of the attack description in a high-dimensional space. The motivation behind this step is to enable comparison with existing Sigma rule embeddings, by representing both the attack descriptions and Sigma rules in the same vector space.

Rule Embedding Construction:

AUTOSIGMA uses an up-to-date collection of Sigma rules, drawn from the latest SigmaHQ³ repository. To organize this repository for efficient search, we build a graph-based representation of those Sigma rules. This representation acts as an indexing method that accelerates rule retrieval during the next stage of AUTOSIGMA. In SigmaHQ repository, rules are already arranged hierarchically into categories such as *application*, *cloud*, *linux*, and *macos*, each containing multiple subcategories including *linux* $\rightarrow$ *auditd*, *file_event*, *process_creation* and *cloud* $\rightarrow$ *aws*, *azure*, *gcp*, *m365*. We convert this directory structure into a lightweight graph in which top-level categories form the first layer, subcategories form the second layer, and individual Sigma rules form the third layer. This abstraction enables efficient traversal and indexing of the rule set during the next stage. Because the SigmaHQ repository is continuously updated, AUTOSIGMA automatically fetches the most recent set of Sigma rules whenever updates are available. It then incorporates these into its knowledge base, ensuring that even cutting-edge detection techniques are considered during rule generation. Deprecated rules are pruned and new rules are indexed without manual intervention, enabling AUTOSIGMA to build on top of the recent rule set. We leverage this repository to find existing Sigma rules that closely match the attack step in question. If a relevant rule is found, we refine and build upon it. Otherwise, we use the rules as one-shot learning examples for the LLM.

³SigmaHQ: <https://github.com/SigmaHQ/sigma>

Similar Rule Discovery:

The repository of Sigma rules serves as the foundation for the similarity matching process. With every rule indexed and embedded, AUTOSIGMA can perform rapid searches to find which existing rules are most relevant to a given attack test. Moreover, because the repository is kept current, the framework ensures that it is comparing against the latest known detection patterns. This means the output rules will align closely with community-curated detection techniques and will not reinvent rules that already exist in the repository. In order to achieve this, we compute BERTScore [63] between an attack step embedding and Sigma rule embeddings to measure how closely a given attack description matches the scope of any known detection rule. The reason why we use BERTScore instead of other similarity scores, such as cosine similarity or BLEU [64], is that this method allows us to capture not only exact word matches but also the overall meaning, fluency, and order of the output, making it more effective than traditional n-gram-based metrics like BLEU. This embedding-based approach allows the system to efficiently scan a large corpus of rules and pinpoint those that are contextually most similar to the new attack scenario.

Definition 3. Let $\mathcal{R}$ denote the set of Sigma rules in the external repository used by AUTOSIGMA. The template matching component associates each attack step with its most relevant Sigma rule templates and is formalized as $f_{tm} : \mathcal{A} \rightarrow 2^{\mathcal{R}}$, where for every $a \in \mathcal{A}$, the image $f_{tm}(a) = T_a \subseteq \mathcal{R}$ is a non-empty finite set of candidate Sigma rule templates found from SigmaHQ repository, that are most similar to a according to the embedding-based similarity metric.

3.3.5 Rule Generation

In the final phase, AUTOSIGMA takes the candidate Sigma rule templates identified in the previous stage and refines them into high-fidelity detection rules tailored to the specific attack scenarios.

Rule Generator:

The rule generation component is designed to transform enriched attack tests into high-quality, executable Sigma rules. The primary goal of this stage is to take the best-matching Sigma templates (see section 3.3.4) and enrich them for the given attack description while ensuring the resulting rule

meets several key criteria. It must accurately reflect the attack techniques described in the source reports and cover all relevant IoCs along with the associated adversary behaviors. At the same time, the rule should maintain strong detection capabilities with minimal false positives, and it must strictly adhere to the Sigma rule format for seamless integration with SIEM platforms. To fulfill these objectives, AUTOSIGMA integrates LLM-as-a-Judge mechanism [57] in which two LLMs work in an adversarial collaboration to iteratively refine the Sigma rules being generated. Details are provided in Appendix A.

LLM-as-a-Judge:

The rule generation phase consists of two distinct LLMs, they can be the same underlying LLMs or they can be two different LLMs, interacting with each other:

1. Rule Generator (LLM-1): This first LLM (the “rule generator”) takes the retrieved Sigma rule template along with the context of the attack step and generates an initial Sigma rule tailored to the scenario. Essentially, LLM-1 fills in or modifies the template’s fields (such as the rule title, description, detection section filters, and tags) using the information from the attack step. The output at this stage is a draft rule hypothesized to detect the described attack.

2. Rule Validator (LLM-2): The second LLM acts as a validator or *judge*. It examines the proposed Sigma rule from LLM-1 and evaluates it against a set of predefined quality checks:

- Does the rule accurately detect the described attack technique and behaviors?
- Is the detection logic valid, sensible, and free of obvious gaps or mistakes (*e.g.*, correct field names, appropriate use of conditions)?
- Is the rule properly formatted according to YAML and Sigma specifications?

Based on this evaluation, LLM-2 provides a quantitative score or verdict on the rule’s quality and generates structured feedback highlighting what needs improvement. For example, the validator might point out that a certain condition is too broad (risking false positives) or that a required field is missing or misformatted. The interplay of these two models — one generating and the other validating — forms an adversarial learning loop. LLM-1 and LLM-2 do not operate in isolation but rather communicate through the intermediate Sigma rule drafts and feedback.

Iterative Refinement Process: Using the generator–validator setup above, AutoSigma iteratively refines each Sigma rule through multiple rounds, as follows:

1. **Initial Draft:** LLM-1 generates an initial enriched rule based on the attack description and the selected template (as described in section 3.3.5).
2. **Validation and Scoring:** LLM-2 evaluates this draft, checking its accuracy, logic, and format, and returns a score along with feedback on how to improve the rule.
3. **Feedback-Driven Revision:** LLM-1 incorporates LLM-2’s feedback and revises the Sigma rule, for example, by tightening detection conditions, adding missing IoCs, or correcting format errors.
4. **Re-Validation:** The updated rule is sent back to LLM-2 for another round of evaluation.
5. **Repeat Until Optimal:** Steps 2–4 repeat, with the rule steadily improving each time, until the validator LLM judges that the rule meets the desired quality threshold. In practice, this loop converges after a few iterations, resulting in a rule that LLM-2 can no longer fault on the predefined criteria.

This refinement loop continues until the Sigma rule is both syntactically valid and semantically robust for detection. We also define a maximum limit of 5 iterations for the LLM-as-a-Judge mechanism to prevent the model from getting stuck in an endless refinement loop. Empirically, we realize that the improvements plateau after 5 iterations. Some errors cannot be caught and improved by LLM, even if we iterate more often on the rules.

Definition 4. Let σ be the set of syntactically valid Sigma rules expressible in the Sigma specification. The rule generation component takes an enriched attack step together with its candidate templates and returns one or more validated Sigma rules. It is modeled as $f_{rg} : \{(a, T) \mid a \in \mathcal{A}, \subseteq \mathcal{R}\} \rightarrow 2^\sigma$, such that for each attack step $a \in \mathcal{A}$ and the associated template set $T_a = f_{tm}(a)$ we obtain a finite set of rules $f_{rg}(a, T_a) = R_a \subseteq \sigma$, where R_a contains the Sigma rules produced by the LLM-as-a-Judge iterative refinement loops that are both syntactically valid and semantically aligned with a .

3.4 Implementation

We build a Proof of Concept (PoC)⁴ using Python 3 programming language. Our experiments were conducted on Intel Xeon E312xx, 6 vCPU @ 2.69Ghz, with 32GB RAM running Ubuntu 20.04. Further details regarding the PoC are provided in Appendix B.

3.4.1 LLM Models

There are generally two methods to work with LLMs: (i) using a cloud-based LLM, trained and deployed by a third-party, or (ii) using a locally hosted LLM. Each approach has its own advantages and disadvantages. Cloud-based LLM access does not require extensive computational resources and simplifies request handling; however, it is often associated with usage costs, and concerns remain about data confidentiality and the possible influence of concurrent API requests on output quality. Running LLMs locally ensures data privacy and eliminates risks associated with multi-tenant environments, but demands significant computing resources. Without sufficient hardware, one must use lighter LLM variants, which may negatively impact performance. For our evaluation, we used both methods:

- *Local-based LLM*: We employed *Lily-Cybersecurity-7B-v0.2*⁵, a fine-tuned variant of Mistral-7B-Instruct-v0.2, specifically trained for cybersecurity tasks. The training encompassed a broad spectrum of cybersecurity domains, including but not limited to APT management, malware analysis, incident response, and secure software development lifecycle. We configured the model with a temperature of 0.1 to minimize randomness and a maximum token length of 512 to ensure concise and relevant outputs.
- *Cloud-based LLM*: We used *ChatGPT-4o-mini*⁶ and *Llama-3.3-70b*⁷ through OpenAI and Llama's APIs. We also set the temperature parameter to 0.1 and the maximum token length to 512 to maintain consistency in output quality and length.

⁴AutoSigma's Demo: <https://youtu.be/iSr6IurQ6BM>

⁵Lily Cybersecurity Model: <https://huggingface.co/segolilylabs/Lily-Cybersecurity-7B-v0.2>

⁶ChatGPT 4o-mini: <https://platform.openai.com/docs/models/gpt-4o-mini>

⁷Llama-3.3-70B: <https://huggingface.co/meta-llama/Llama-3.3-70B-Instruct>

3.4.2 Datasets

Threat Reports: We use threat reports from the APTNotes repository⁸, which gathers different threat intelligence from different vendors (*e.g.*, Mandiant, CrowdStrike). Given the lack of a comprehensive dataset, we focused only on APT41, APT28, and APT29, yet the solution can work with any report. Appendix C provides a summary of the CTI reports used in our evaluation. To further evaluate different solutions, we also used the dataset released by LLMCLOUDHUNTER [20], comprising 20 cloud-security blogs (Azure/AWS/GCP) paired with their manually crafted Sigma rules.

Knowledge Base: AUTOSIGMA incorporates two external knowledge bases. The NVD, which provides a comprehensive repository of CVEs. We utilize this database to extract relevant information about identified vulnerabilities. The second is the official SigmaHQ repository, which contains more than 3,600 rules spanning over 60 detection domains. These rules are distributed across several major categories, including but not limited to, operating system domains (*e.g.*, Windows, Linux, macOS), cloud environments (*e.g.*, AWS, Azure, GCP, M365) and more. This diversity reflects the breadth of the Sigma rule ecosystem.

3.5 Evaluation

We adopt three research questions that jointly assess AUTOSIGMA quantitatively and qualitatively: (i) benchmarking against state-of-the-art solution; (ii) cross-APT evaluation against LLM models, and (iii) design-choice and robustness studies isolating the effects of the LLM-as-a-Judge loop and model variation and sampling temperature.

RQ1. How effective is AUTOSIGMA in generating high-quality and behavior-aligned Sigma rules compared to a state-of-the-art solution?

We evaluate AUTOSIGMA, using LLMCLOUDHUNTER dataset, in two configurations AUTOSIGMA using cloud-based LLMs and AUTOSIGMA using a Local LLM, and compare against LLMCLOUDHUNTER.

⁸APTNotes repository: <https://github.com/aptnotes/data/>

All systems are assessed with seven complementary metrics: (i) *No. Rules*: number of generated Sigma rules per report; (ii) *Rule Validity*: percentage of rules that are syntactically valid and convertible to SIEM queries (e.g., Splunk/ELK) using sigma-cli⁹; (iii) *Condition Accuracy*: focuses on the correctness of the condition fields, which specify the relationship between various selection fields. (iv) *Rule Relevancy*: semantic similarity between each report and its generated rule set via BERTScore [65]. (v) *Partial Coverage*: number of generated rules that match at least one of the same MITRE ATT&CK techniques as a ground truth rule; (vi) *Full Coverage*: number of generated rules whose detection logic is equivalent to a ground truth rule; (vii) *Newly Discovered*: number of new rules generated by the model but not present in the ground truth; The results are summarized in Table 3.1 and discussed in the following sub-research questions.

Table 3.1: Average results of AUTOSIGMA against LLMCLOUDHUNTER using cloud-security blogs.

	AUTOSIGMA (Cloud-based OpenAI)	AUTOSIGMA (Local-based Lily)	LLMCloudHunter
No. Rules	21	25	8
Rule Validity	100	77.94	94
Condition Accuracy	100	100	100
Rule Relevancy	80.4	77.7	76.4
Partially Covered	97.40	84.41	80.9
Fully Covered	83	74.93	74
Newly Discovered	12	11	2

RQ1.1. Are the generated rules syntactically correct, semantically consistent with the input reports? Results for total number of rules and rule validity are shown in Table 3.1. As we can see, AUTOSIGMA in both cloud-based and local-based, outperformed the state-of-the-art solution. Due to the fact that LLMCLOUDHUNTER tends to generate the rules based on the telemetry information, such as IoCs mentioned in the report, this limits the overall number of rules produced. For rule validity, the cloud-based version of AUTOSIGMA achieves a score of 100%, meaning every generated rule is syntactically valid and can be directly converted into a SIEM query. The local version, while producing a slightly lower validity percentage compared to LLMCLOUDHUNTER, primarily because it uses a lighter-weight LLM, still maintains a higher count of valid rules than LLMCLOUDHUNTER due

⁹SigmaCLI: <https://github.com/SigmaHQ/sigma-cli>

to its larger output volume. This difference between the cloud-based and local-based configurations stems from their underlying LLM capabilities and generation precision, as further discussed in Appendix D. Furthermore, the condition accuracy metric shows that all generated rules, across all evaluated models, achieve 100% correctness in their condition fields. This indicates that the logical structure of each rule faithfully captures the intended detection behavior and can correctly operationalize the threat activity described in the reports.

RQ1.2. Do the generated rules correspond to behaviors explicitly described in the original reports? In this experiment, all models generate rules that remain semantically aligned with their corresponding CTI reports. However, as the rule relevancy (Table 3.1) shows that both cloud-based and local-based versions of AUTOSIGMA, achieve higher semantic similarity scores (80.4% and 77.7%, respectively) than the state-of-the-art solution (76.4%). This demonstrates AUTOSIGMA’s strength in maintaining contextual coherence throughout its Attack Scenario Extraction and Enrichment. Even with a higher number of generated rules, which typically increases the risk of semantic drift, AUTOSIGMA preserves alignment with the original text due to its contextual analysis and LLM-as-a-judge components, ensuring the final outputs remain faithful to the report’s intent.

RQ1.3. To what extent do the rules generated capture the threat behaviors and MITRE ATT&CK techniques documented in the reports? As shown in Table 3.1, both versions of AUTOSIGMA stand ahead of LLMCLOUDHUNTER. In terms of partially covered rules, AUTOSIGMA has covered 97.4% and 84.4%, versus 80.9% for LLMCLOUDHUNTER. For fully covered rules, AUTOSIGMA has 83.0% compared to 74.0% obtained by LLMCLOUDHUNTER. This is due to the fact that LLMCLOUDHUNTER tends to focus on telemetry information such as IoCs when generating Sigma rules, which limits the overall number of rules produced. Moreover, LLMCLOUDHUNTER is specifically designed to filter cloud-based threat scenarios and therefore only generates cloud-based rules. Overall, all models perform reasonably well since they rely on the MITRE ATT&CK technique extraction at some stage of their process. However, AUTOSIGMA demonstrates superior coverage by 8%, because its design does not have any domain constraint filters. Unlike LLMCLOUDHUNTER, AUTOSIGMA identifies techniques across diverse contexts, resulting in broader and more comprehensive coverage.

RQ1.4. Can we identify new or previously uncovered Sigma rules? The newly discovered

metric highlights the capacity of a model to uncover attack behaviors not explicitly reflected in the ground truth. Manual rule creation often overlooks such subtle variations or decomposed steps. As shown in Table 3.1, AUTO SIGMA achieves an average newly discovered rule of 12 compared to LLM CLOUD HUNTER of 2, as AUTO SIGMA employs attack scenario extraction and decomposition components, which break complex attack narratives into simpler attacks that are more effectively expressed as Sigma rules. AUTO SIGMA not only focuses on telemetry information described in the threat report, but also leverages external knowledge bases. This process enables AUTO SIGMA to discover a greater number of novel rules (12 and 11 on average) compared to LLM CLOUD HUNTER.

RQ2. How does AUTO SIGMA perform relative to LLM baseline models in generating Sigma rules for APT reports?

To evaluate the performance of AUTO SIGMA against LLM models, we defined four metrics: (i) *Coverage*: evaluate the content retention from the report at two levels: (a) *IoC coverage* by checking if IoCs mentioned in the report have been covered by the generated rules, and (b) *MITRE coverage* by checking the covered techniques from the report and the generated rules. For this experiment we used Threat2Mitre [66] as our baseline; (ii) *Relevancy*: Semantic alignment between each report and its generated rule set using BERTScore embeddings; (iii) *Validity*: rules that are syntactically valid and convertible to SIEM queries (*e.g.*, Splunk/ELK) using sigma-cli; and (iv) *Adaptability*: Pairwise BERTScore similarities among *inputs* (reports) and among *outputs* (rules). It is important to mention that LLM CLOUD HUNTER is designed for cloud-specific content and does not produce any rules if the content is not related to cloud. Thus, we excluded it in this evaluation since our focus is on APT reports.

RQ2.1. Does AUTO SIGMA achieve higher coverage of described behaviors? Table 3.2 shows the number of Sigma rules generated per report. On average, ChatGPT-off-the-shelf generates only 6 rules per report, which is often insufficient given the sophistication and multi-stage nature of APT attacks. In contrast, AUTO SIGMA, under different configurations, because its system design, specifically its attack scenario decomposition component, consistently produces a higher number of rules, around 28 for local models and 24 for Cloud-based models.

To further validate the generated rules, we manually extracted the IoCs from the CTI reports and

Table 3.2: Number of Sigma rules generated per APT41 reports.

Report	Cloud-Based LLMs		Local-Based LLMs	Baseline
	AutoSigma Llama	AutoSigma OpenAI	AutoSigma Lily	ChatGPT off-the-shelf
Mandiant#1	18	21	25	6
Mandiant#2	14	19	22	6
Mandiant#3	30	32	36	5
Mandiant#4	25	28	30	7
TrendMicro	24	24	28	7
Avg.	22	25	28	6

checked whether each IoC appears in at least one of the generated Sigma rules.

Table 3.3 reports the coverage results using different models. In a Cloud-based LLM environment, AUTO SIGMA achieves near-total coverage of IoCs; on average, it covers 95.6% and 95.2% of all IoCs reported in the underlying CTI report. For example, it covers 23 of 25 IoCs (92%) for Mandiant#1, 20/21 (95%) for Mandiant#2, and 29/32 (91%) for TrendMicro (APT41). In contrast, ChatGPT-off-the-shelf captures only a few IoCs (*e.g.*, 3/25 in Mandiant#1, 5/32 in APT41), for an average of just 17.6%. These results indicate that AUTO SIGMA fully incorporates the reported IoCs (several reports reach 100% coverage), whereas the unguided LLM misses the majority. In Local-based LLMs, AUTO SIGMA covers only about half of the IoCs, coverage ranges from 12/25 (48%) for Mandiant#1 to 8/10 (80%) for Mandiant#4, with an average of 56.0%. Appendix D provides a discussion on this gap and the difference between local and cloud-based LLM. The trend identified for APT41 holds consistently for the other APTs as well. Across APT28 and APT29, AUTO SIGMA significantly improves IoC and MITRE coverage for both cloud-based and local LLMs, while also achieving higher semantic relevancy than the ChatGPT-off-the-shelf baseline. Detailed per-APT results are reported in Appendix E.

We further assessed the technique-level coverage by extracting the set of MITRE ATT&CK techniques mentioned in each CTI report and comparing it to those appearing in the generated Sigma rules. To automate this mapping, we employ the Threat2Mitre [66], which uses AI-driven language models to process natural-language threat descriptions and link them to the MITRE ATT&CK

Table 3.3: IoC-level coverage comparison between APT41 reports and generated Sigma rules.

Report	Ground Truth	Cloud-Based LLMs		Local-Based LLMs	Baseline
		AutoSigma Llama	AutoSigma OpenAI	AutoSigma Lily	ChatGPT off-the-shelf
Mandiant#1	25	23 (92%)	23 (92%)	12 (48%)	3 (12%)
Mandiant#2	21	19 (90%)	20 (95%)	9 (43%)	4 (19%)
Mandiant#3	14	14 (100%)	14 (100%)	7 (50%)	3 (21%)
Mandiant#4	10	10 (100%)	10 (100%)	8 (80%)	2 (20%)
TrendMicro	32	30 (94%)	29 (91%)	19 (59%)	5 (16%)
Avg.	—	95.2%	95.6%	56.0%	17.6%

framework.

We can see in Table 3.4 that AUTO SIGMA outperforms the baseline in terms of MITRE ATT&CK coverage. With the cloud-based model, AUTO SIGMA OpenAI and AUTO SIGMA Llama achieve an average coverage of 91.2% and 90.8% of the techniques extracted from the reports, respectively, whereas ChatGPT-off-the-shelf covers only about 20.2%, due to the contextual analysis stage of the solution. Using the local LLM yields a lower but still high coverage of 70.8% for AUTO SIGMA. These percentages are derived from the per-report counts, for example, in the Mandiant#3, Threat2Mitre finds 21 techniques in the report, and AUTO SIGMA’s cloud-based rules cover 19 of them when using Llama LLM, and 18 when using OpenAI LLM (90% and 86% respectively), whereas ChatGPT-off-the-shelf’s rules cover only 3 (21%). Overall, AUTO SIGMA’s Sigma rules retain the vast majority of the techniques mentioned in the reports, demonstrating that the system preserves the original threat intelligence. Importantly, AUTO SIGMA also infers new additional techniques beyond those explicitly mentioned in the reports. In total, across the five APT reports, AUTO SIGMA’s cloud-based rules introduce 91 new MITRE techniques (an average of 18 additional techniques per report), whereas ChatGPT-off-the-shelf adds only 12 new techniques overall.

This enrichment is driven by AUTO SIGMA’s use of external knowledge bases and attack decomposition. By consulting structured MITRE ATT&CK data and decomposing high-level attack behaviors into finer steps, AUTO SIGMA can infer implicit TTPs that the original report did not enumerate. This observation is also witnessed across all APTs, reported in Appendix E and Table E.1.

Table 3.4: MITRE ATT&CK-level coverage comparison between APT41 reports and generated Sigma rules.

Report	Ground Truth	Cloud-Based LLMs						Local-Based LLMs			Baseline		
		AutoSigma Llama			AutoSigma OpenAI			AutoSigma Lily			ChatGPT off-the-shelf		
		Total	Covered	New	Total	Covered	New	Total	Covered	New	Total	Covered	New
Mandiant#1	8	16	8 (100%)	8	15	8 (100%)	7	10	6 (75%)	4	4	2 (25%)	2
Mandiant#2	12	14	10 (83%)	4	17	11 (92%)	6	9	7 (58%)	2	5	2 (17%)	3
Mandiant#3	21	29	19 (90%)	10	30	18 (86%)	12	20	14 (67%)	6	5	3 (14%)	2
Mandiant#4	13	24	13 (100%)	11	20	12 (92%)	8	16	10 (77%)	6	6	4 (31%)	2
TrendMicro	22	24	18 (81%)	6	23	19 (86%)	4	20	17 (77%)	3	5	3 (14%)	2
Avg.	–		90.8%			91.2%			70.8%			20.2%	

Moreover, we examined the relevancy of generated rules, by checking whether the generated Sigma rules reflect the semantic content of the input threat reports. To quantify this, we compute BERTScore [65] between each CTI report and its corresponding Sigma rules. BERTScore maps tokens to contextual embeddings and computes pairwise cosine similarities, yielding a score $[0 - 1]$ that correlates with semantic alignment. A higher BERTScore indicates that the rule’s language captures more of the report’s meaning.

Table 3.5 presents the per-report BERTScores for all models for APT41. Both cloud-based versions of AUTO SIGMA achieve very similar results, with approximately the same average across reports. This shows that AUTO SIGMA is a model-agnostic solution and the variation of the underlying LLM has minimal effect on the relevancy of the generated rules. In each APT report, AUTO SIGMA’s output consistently achieves the highest relevancy, outperforming the baseline. For example, in Mandiant#2, the scores are 0.778 (AUTO SIGMA OpenAI) vs 0.718 (ChatGPT-off-the-shelf). Averaging across all five reports, AUTO SIGMA attains an average BERTScore of 0.764 and 0.765, compared and 0.718 for ChatGPT-off-the-shelf.

AUTO SIGMA’s generated rules include key phrases and contextual details from the report that ChatGPT-off-the-shelf sometimes omits or paraphrases less accurately. We also observed the same pattern when using a local LLM instead of a cloud-based model. Even though absolute scores are slightly lower, AUTO SIGMA using local LLM leads with a mean of 0.738 and 0.718 for ChatGPT-off-the-shelf.

RQ2.2. Are the rules produced by AUTO SIGMA syntactically valid and executable? We define a generated Sigma rule to be *valid* if it is syntactically and operationally correct. In practice,

Table 3.5: BERTScore comparison between the input CTI report and the generated Sigma rules.

Report	Cloud-Based LLMs		Local-Based LLMs	Baseline
	AutoSigma Llama	AutoSigma OpenAI	AutoSigma Lily	ChatGPT off-the-shelf
Mandiant#1	0.766	0.762	0.722	0.709
Mandiant#2	0.779	0.778	0.748	0.718
Mandiant#3	0.749	0.752	0.723	0.701
Mandiant#4	0.771	0.775	0.745	0.729
TrendMicro	0.759	0.755	0.753	0.732
Avg.	0.765	0.764	0.738	0.718

this means the rule must be compiled without errors, and its detection logic should detect at least one relevant malicious event.

Table 3.6 shows the generated Sigma rules validity results. We can see that AUTO SIGMA cloud-based, achieves 100% validity, where every rule generated was valid (100% across all tested reports).

In the local-based setting, rule validity is at 76.2% and it is lower compared to the baseline, because a lighter model is more prone to syntax slips. But, if we consider the number of valid rules that local-based AUTO SIGMA has generated, we have a larger number of rules compared to the baseline. For example, in Mandiant#4, even though the validity percentage of AUTO SIGMA is lower, it has 25 valid and ready-to-use rules, compared to the 6 valid rules generated by ChatGPT-off-the-shelf.

RQ2.3. Can AUTO SIGMA better adapt to unseen report structures? Adaptability measures whether AUTO SIGMA generates contextually distinct Sigma rules for different input reports, *i.e.*, whether semantically different threat reports produce correspondingly different rules rather than generic hallucinations. To assess this, we compute pairwise semantic similarity using BERTScore for (i) the original APT reports, (ii) the Sigma rules generated by AUTO SIGMA, and (iii) the Sigma rules generated by ChatGPT-off-the-shelf. Fig. 3.4a shows the semantic similarity among the original CTI reports. We see a wide range of values (≈ 0.28 – 0.72 off the diagonal), confirming that the reports differ substantially in content. Figs. 3.4b–3.4d show that AUTO SIGMA has much lower off-diagonal similarity overall (means ≈ 0.43 – 0.50) and values dropping to ≈ 0.17 in some

Table 3.6: Comparison of rule validity generated by different models across different APT41 reports.

Report	Cloud-Based LLMs		Local-Based LLMs	Baseline
	AutoSigma Llama	AutoSigma OpenAI	AutoSigma Lily	ChatGPT off-the-shelf
Mandiant#1	18/18 (100%)	21/21 (100%)	18/25 (72%)	6/6 (100%)
Mandiant#2	14/14 (100%)	19/19 (100%)	16/22 (73%)	6/6 (100%)
Mandiant#3	30/30 (100%)	32/32 (100%)	28/36 (78%)	5/5 (100%)
Mandiant#4	25/25 (100%)	28/28 (100%)	25/30 (83%)	6/7 (86%)
TrendMicro	24/24 (100%)	24/24 (100%)	21/28 (75%)	7/7 (100%)
Avg.	100.0%	100.0%	76.2%	97.2%

cases. The generated rules from distinct reports are clearly less similar to each other, reflecting the input variability. For example, AUTO SIGMA’s cloud-based configurations yield low similarity for pairs of reports that are very different (near 0.17–0.22) and higher similarity only when the original reports are themselves more alike. AUTO SIGMA using a local-based LLM, shows a similar pattern. These results demonstrate that AUTO SIGMA’s Sigma rules are contextually tailored, showing that it preserves semantic differences between reports in the output. By contrast, Fig. 3.4d shows that ChatGPT-off-the-shelf exhibits uniformly high similarities among rule outputs. Most off-diagonal entries are above ≈ 0.5 (mean ≈ 0.60 , with values up to 0.84). This indicates that ChatGPT-off-the-shelf tends to generate very similar (often repetitive) Sigma rules regardless of the input report, failing to differentiate between distinct reports.

RQ3. How robust is AUTO SIGMA to different design choices (such as the use of LLM-as-a-judge, model variation and sampling temperature), and how do these choices influence the validity, relevancy and stability of generated sigma rules

RQ3.1. Ablation study: What is the impact of using an LLM as an automated evaluator on rule validity and rule relevancy? In this study, we compare with/without the iterative judge loop on validity. This isolates the benefit of automatic critique–refine cycles. As shown in Table 3.7, the cloud-based models equipped with the LLM-as-a-judge mechanism achieve the highest validity,

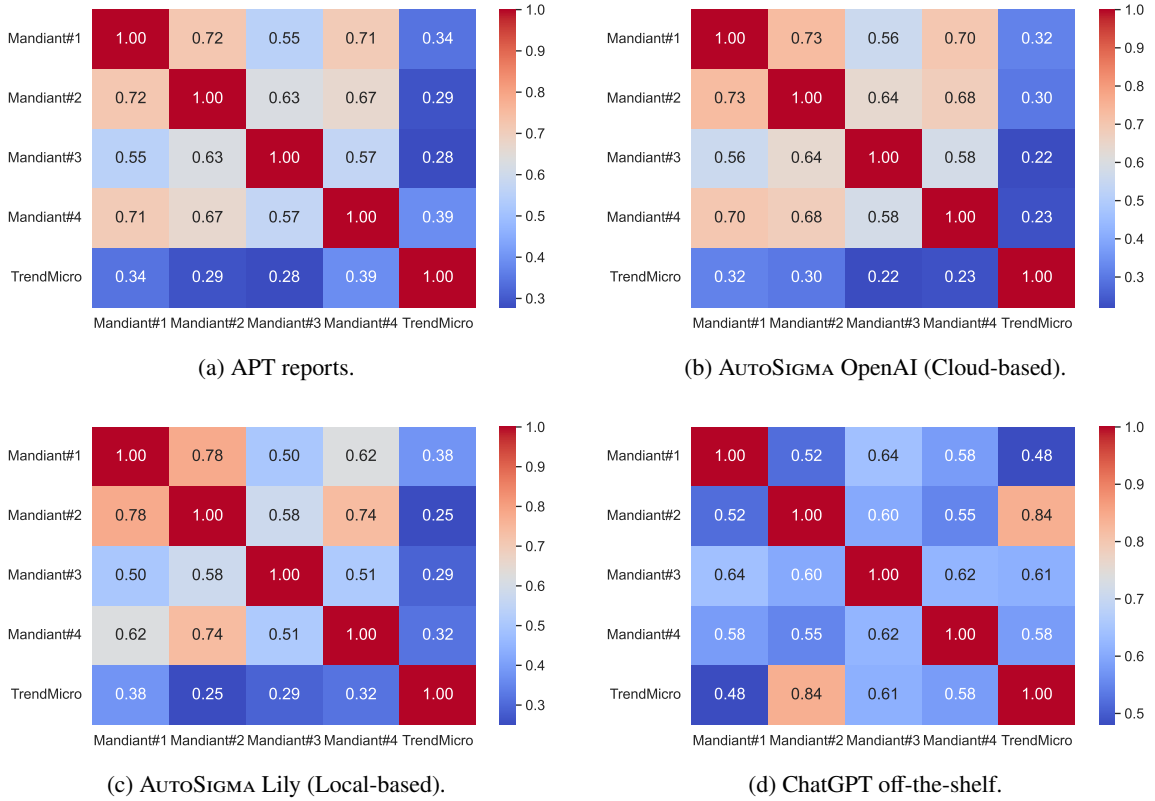


Figure 3.4: Comparison of semantic similarities between pairs of inputs and pairs of generated Sigma rules across different APT41 reports.

with 100% of their generated outputs being valid and directly convertible into SIEM rules. In contrast, models that do not have the LLM-as-a-judge component exhibit lower validity ratios. For example, in the TrendMicro report, the could-based AutoSIGMA (OpenAI) without the LLM-as-a-judge generated 21 valid rules out of 24 total rules, whereas using LLM-as-a-judge increases this to 100%. In the local setting, a similar trend is observed, the version without LLM-as-a-judge achieves an average validity of 47.5% across all APTs, while integrating the judge increases this to 71.7%, representing an improvement of 24.2%.

RQ3.2. Design choice study: How does the sampling temperature influence the balance between rule diversity and hallucination frequency? Hallucination is a major challenge in LLM-based solutions. Although AutoSIGMA uses LLM-as-a-judge to mitigate the risk of generating hallucination, there is currently no standardized metric for quantifying hallucinated content in this context. Therefore, we analyzed behaviors that affect the likelihood of hallucination and present the measures taken in AutoSIGMA to reduce this likelihood.

Table 3.7: Comparison of rule validity by different models across different APT41 reports.

Report	Cloud-Based LLMs			Local-Based LLMs		
	AutoSigma Llama	AutoSigma Llama	AutoSigma OpenAI	AutoSigma OpenAI	AutoSigma Lily	AutoSigma Lily
	(w/ Judge)	(w/o Judge)	(w/ Judge)	(w/o Judge)	(w/ Judge)	(w/o Judge)
Mandiant#1	18/18 (100%)	16/18 (89%)	21/21 (100%)	18/21 (86%)	18/25 (72%)	13/25 (52%)
Mandiant#2	14/14 (100%)	12/14 (86%)	19/19 (100%)	17/19 (89%)	16/22 (73%)	10/22 (45%)
Mandiant#3	30/30 (100%)	24/30 (80%)	32/32 (100%)	26/32 (81%)	28/36 (78%)	20/36 (56%)
Mandiant#4	25/25 (100%)	21/25 (84%)	28/28 (100%)	26/28 (93%)	25/30 (83%)	16/30 (53%)
TrendMicro	24/24 (100%)	21/24 (88%)	24/24 (100%)	21/24 (88%)	21/28 (75%)	12/28 (43%)
Avg.	100.0%	85.4%	100.0%	87.4%	76.2%	49.8%

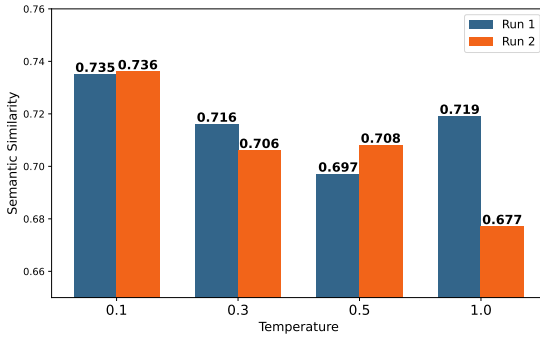


Figure 3.5: Effect of temperature on semantic similarity.

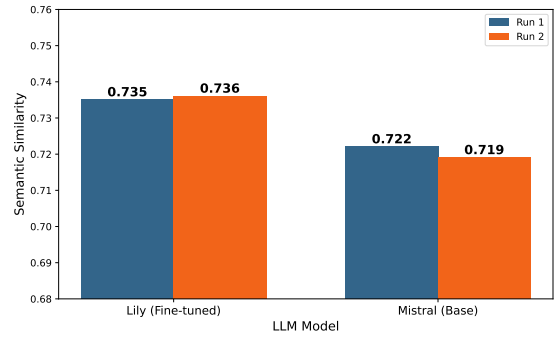


Figure 3.6: Effect of model fine-tuning on hallucination.

Temperature [67] is a key controllable parameter in LLM sampling that controls randomness and diversity. Higher temperatures tend to increase linguistic variation and “creativity” but also introduce instability and hallucinated logic. To evaluate its impact, we conducted an experiment using one representative APT report and ran AUTO SIGMA twice per temperature value $T = \{0.1, 0.3, 0.5, 1.0\}$. For each run, we computed the semantic similarity using BERTScore between the generated rules and the original CTI report. As shown in Fig. 3.5, the outputs at $T = 0.1$ remain consistent across runs, while higher temperatures cause semantic divergence. This confirms that maintaining a low temperature (0.1 in our main experiments) provides stable, low-variance results and effectively suppresses hallucinated content.

RQ3.3. Does domain-specific fine-tuning reduce hallucination and improve coverage compared to prompt-only generation? Fine-tuning the base LLM with domain-specific data can further improve the factual grounding and consistency of generated rules. To validate this, we compared two

variants of AUTO SIGMA using the same temperature setting ($T = 0.1$): one with the cybersecurity fine-tuned model *Lily-Cybersecurity-7B-v0.2* and another using a generic Mistral LLM. As shown in Fig. 3.6, the fine-tuned Lily model achieves consistently higher semantic similarity across runs, confirming its reduced hallucination behavior. This observation aligns with prior work such as CyberPalAI [68].

3.6 Discussion

The evaluation of AUTO SIGMA highlights its strong performance in generating high-quality, deployable Sigma rules. However, beyond reporting numerical metrics, it is essential to critically assess the solution’s design and identify areas where improvements can be made. A promising direction for enhancing robustness involves combining generative reasoning with rule-based validation logic, such as leveraging structured grammars, detection logic templates, and constraints informed by known adversary behaviors. This would enable more principled rule synthesis that does not depend solely on LLM fluency. Additionally, the current rule generation loop is generative at every iteration. Moving toward a guided refinement process, where intermediate rule candidates are evaluated not only by another model but also by static analyzers or predefined criteria, could reduce hallucinations and enforce stronger alignment with security semantics. A separate concern is the execution time of the system. Because AUTO SIGMA operates as a multi-stage pipeline with enrichment, template retrieval, and iterative validation, the overall latency can be non-trivial. While this is acceptable for offline threat hunting use cases—where rules are generated as part of periodic analysis—it becomes a limitation in time-sensitive environments. Although the pipeline was not designed for real-time deployment and it is designed for an offline deployment. Improving the efficiency of the validation loop and introducing caching or early-exit strategies could significantly reduce processing delays and make the system more responsive. Despite these performance trade-offs, it is worth emphasizing that the rule synthesis process only needs to run once per report and does not interfere with real-time alert handling, making this overhead manageable in most hunting workflows.

Chapter 4

ARMS: Automated Rule Mutation Solution

4.1 Motivation and Problem Statement

Motivation: Security teams rely on Sigma rules to support detection and to address diverse operational contexts. However, the manual creation of Sigma rules is a significant bottleneck. Traditional manual workflows are labor-intensive, error-prone, and require deep domain expertise, making them difficult to scale in the face of a rapidly expanding threat landscape. Despite the growth of open-source repositories like SigmaHQ [48], these collections are inherently reactive. While attackers are dynamic and evolutionary, rules remain static. A rule designed to catch a specific PowerShell-based credential dumping technique becomes obsolete the moment an adversary switches to a Mac OS host or a direct API implementation. Furthermore, copying and pasting rules generated by automated solutions introduces significant operational friction. Most automated solutions generate rules based on the specific telemetry of the organization that was originally attacked. However, every enterprise possesses a unique digital fingerprint, including specific log schemas, network topologies, and legitimate administrative behaviors. Importing generic or external rules without environmental context leads to the deployment of redundant rules that target non-existent assets and the generation of excessive false-positive alerts.

Problem Statement: Current research in automated detection engineering primarily focuses on the

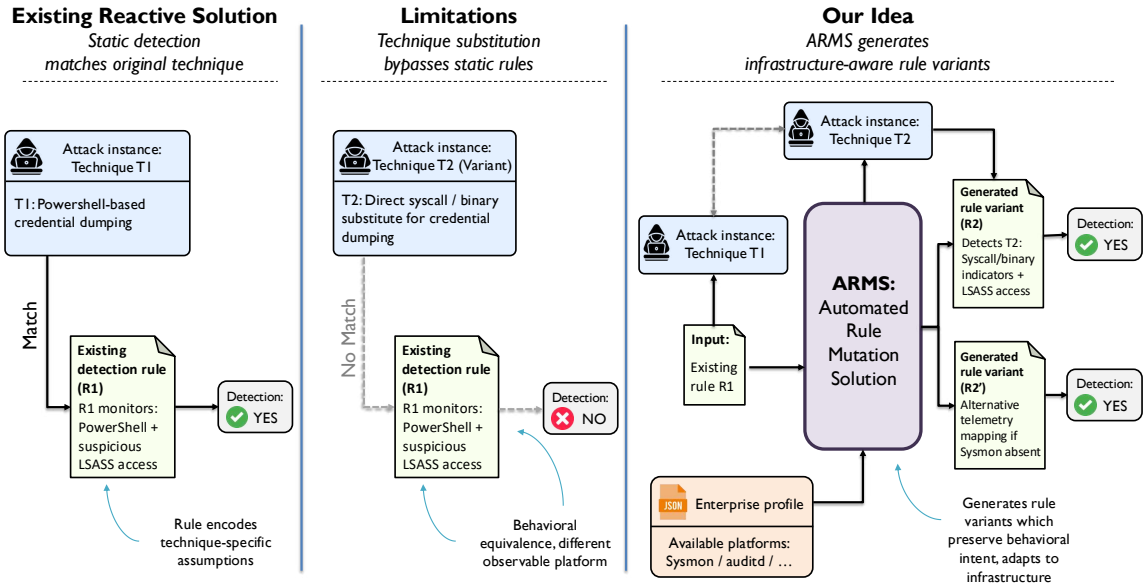


Figure 4.1: Illustration of the problem statement and motivation behind ARMS.

direct generation of structured and static rules [20, 21, 25, 50, 55]. As shown in Figure 4.1 (left and middle sections), this approach is fundamentally limited, since these rules are static. The resulting rules will fail after a technique shift from an adversary. Existing methodologies [20, 21, 25, 50, 55] are largely focused on a mapping from a report to a rule set, neglecting the necessity for generating rule variants that cover the broader behavioral spectrum of a single attack technique.

The right section of Figure 4.1 depicts the idea of ARMS. ARMS is designed to bridge this gap by transitioning from static rule extraction to a proactive, infrastructure-aware variant generation. Instead of relying on static rules which can be evaded after an adversary performs a technique shift, ARMS gets the static rule as input, it identifies the possible technique substitution based on the infrastructure of the environment, and assisted with LLMs, ARMS generates variant detection rules that cover the technique shifts.

Limitation of existing solutions: Previous research has investigated methods to reduce the cost of generating initial detections, including approaches that apply Natural Language Processing (NLP) and Large Language Models (LLMs) to generate detection rule candidates from unstructured sources [20, 21]. Other attempts tend to address the operational cost of large rule sets [25] by optimizing and consolidating rule sets to reduce redundancy and improve efficiency. As shown in Table 2.2, these approaches do not address a fundamental challenge frequently encountered in detection operations:

① maintaining coverage when attackers alter their techniques, ② ensuring that the rules remain valid and compatible with the target environment. Although evasion-aware rule hardening [26] can enhance robustness against bypass patterns, ③ it does not address technique substitution and behavioral drift outside the original rule’s scope.

4.1.1 Contribution and Advantages of ARMS

Contribution: To address these challenges, we propose ARMS, an automated rule mutation solution designed to generate resilient infrastructure-aware detection rule variants. Unlike previous work [20, 21] that focuses on static rule generation, ARMS emphasizes the proactive mutation of detection rules to anticipate emerging threat variants while ensuring validity and compatibility with the target environment. In this paper, we make the following contributions:

- To address ①, ARMS defines a novel three-dimensional space to identify attack variants based on the logic of adversarial equivalence. ARMS identifies technique proximity based on structure (*e.g.*, graph-based relationships), function (*e.g.*, semantic similarity), and environment (*e.g.*, shared prerequisites and exploitable vulnerabilities via CAPEC [69] mappings).
- To address ②, ARMS implements a mechanism to filter and customize detection rules according to an enterprise’s specific infrastructure. By utilizing a structured infrastructure profile of organizational assets, ARMS ensures that generated rules are optimized and relevant to the environment in which they are deployed.
- To address ③, ARMS utilizes Retrieval-Augmented Generation (RAG) [51] and Chain-of-Thought (CoT) reasoning [70] to convert detection logic across different technologies, such as converting Windows-centric Sysmon logic into equivalent Linux auditd or cloud-native signatures.
- We established a robust methodology to assess generated variant rules across four critical research questions: variant fidelity, rule correctness, including syntactic and semantic validity, and infrastructure relevance.

Advantages of ARMS: ARMS provides several advantages for modern Security Operations Centers: (1) it enhances detection resilience by broadening coverage beyond one specific attack, effectively neutralizing an adversary’s ability to evade detection through technique shifts; (2) ARMS significantly reduces the operational overhead of detection by automating the refinement and translation phases that previously required human and manual expert intervention; and (3) ARMS minimizes the noise generated by generic rules by ensuring that every mutation is relevant to the specific infrastructure and telemetry capabilities of the enterprise.

4.2 Threat Model and Assumptions

Scope: In-scope threat model focuses on agile adversaries who achieve the same tactical objective by changing behavior, *i.e.*, substituting techniques to bypass static signatures. We target behavioral variants where the execution path shifts (*e.g.*, using alternative binaries or different scripting methods) rather than simple evasion such as command-line obfuscation. Out of scope are (i) zero-day exploits and vulnerability vectors with no behavioral precedent, since ARMS mutates known behaviors rather than anticipating entirely new exploitation paradigms; (ii) attacks against the generation engine itself (*e.g.*, prompt-injection embedded in input rules); and (iii) trivial syntactic obfuscation (*e.g.*, casing changes, argument reordering), which we treat as the responsibility of log normalization rather than behavioral-mutation logic generation.

Assumptions: We assume the semantic correctness of external knowledge bases (*e.g.*, MITRE ATT&CK and Atomic Red Team), which we treat as ground truth for adversary behavior. We also assume well-formed Sigma rules from trusted repositories [48], including accurate technique metadata tags. Finally, ARMS operates offline on infrastructure definitions rather than live telemetry; it does not require streaming production logs and instead relies on static profiles to infer available data sources.

4.3 ARMS: Automated Rule Mutation Solution

4.3.1 System Overview

Figure 4.2 illustrates the architecture of ARMS. The solution operates in four distinct phases, transforming static detection logic into a dynamic, infrastructure-aware defense posture. The first phase, Rule Ingestion, addresses the syntactic heterogeneity of detection rules. Although Sigma provides a standardized format, individual rule structures often vary. ARMS normalizes these inputs by parsing the raw YAML file into a unified Abstract Syntax Tree (AST) [71]. AST is a hierarchical tree representation of the syntax structure of a given source code or a rule. Using this method, ARMS extracts critical metadata such as detection fields and the underlying attack technique. The second phase, Rule Mutation, serves as the core engine of ARMS and is composed of a three-dimensional proximity logic to identify substitutable techniques. Rather than relying solely on hierarchical relationships, this component searches for semantically equivalent behaviors across different knowledge bases (MITRE ATT&CK, CAPEC) that fit the structural, functional, and environmental proximity of the rule in question. The third phase, Rule Optimization, acts as a strategic filter between the mutation and adaptation steps. Naively generating rules for every possible technique variant leads to computational waste and alert fatigue if the resulting rules are irrelevant to the target environment. To mitigate this, ARMS analyzes the enterprise's specific infrastructure. This agent removes variants that require unavailable telemetry (*e.g.*, discarding Linux-specific variants in a Windows-only environment) before they are passed to the adaptation phase. Finally, the fourth phase in ARMS is the Rule Adaptation, which contains a RAG-based generative agent responsible for translating the syntax of the original rule to match the newly identified technique variants. It consists of multiple components responsible for grounding the Atomic Red Team based on the techniques, finding the closest test case, constructing the CoT-based prompt, generative execution, and lastly, rule validation.

Design Challenges:

Transforming a static signature into a diverse set of behavioral variants introduces several semantic and operational hurdles. We categorize these challenges as follows:

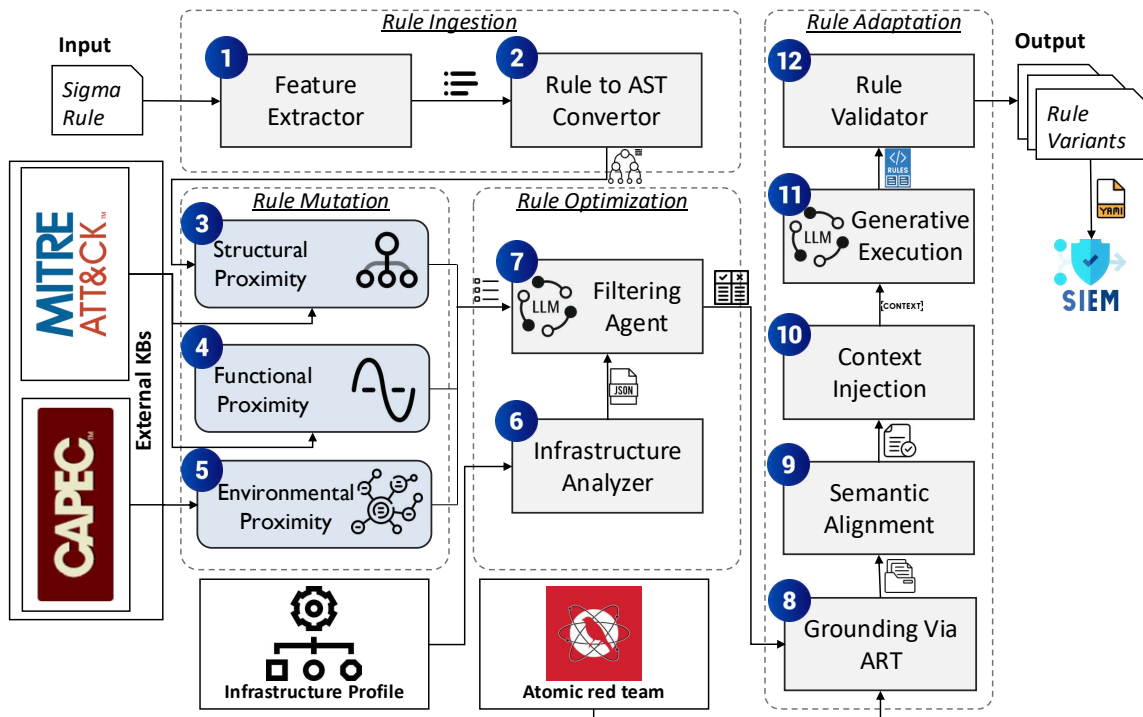


Figure 4.2: High-level overview of ARMS and its components.

- 1 Syntactic Heterogeneity:** While detection languages like Sigma aim for unification, the internal structure of rules often differs significantly based on the author's style and the target log source. A robust system must be able to ingest and normalize these diverse structures without losing the logic's intent.
- 2 Semantic Equivalence Discovery:** A rule is not merely a string of keywords; it represents a specific adversarial procedure. To mutate a rule effectively, the system must determine which alternative techniques achieve the same tactical goal. Identifying these non-obvious, inter-changeable techniques beyond simple parent-child relationships is a complex retrieval problem.
- 3 Cross-Domain Syntax Adaptation:** When a variant technique crosses operating system boundaries (e.g., adapting a PowerShell attack to a Linux environment), the original detection logic becomes invalid. The system must fundamentally rewrite the syntax to match the new environment's CLI and logging standards while preserving the attack's behavioral signature.

- 4 Infrastructure-Aware Optimization:** Indiscriminate generation of variants wastes computational resources and overwhelms security analysts. The challenge lies in optimizing the output to ensure that only deployable, relevant rules which are compatible with the organization's active assets and telemetry, are generated.

Novelty:

ARMS addresses these challenges through a set of novel technical contributions: To address **1**, we propose a Unified AST Ingestion Engine. Unlike regex-based parsers, our approach converts diverse Sigma rules into a standardized Abstract Syntax Tree. This allows ARMS to programmatically manipulate detection logic regardless of the input's stylistic variations. To solve **2**, we introduce a 3-Dimensional Proximity Space. Moving beyond naive hierarchical expansion (e.g., only looking at T1059.001 vs. T1059.002), our engine calculates semantic proximity across three axes: Structural, Functional, and Environmental. This allows the system to identify distant but tactically equivalent techniques that standard lookup tables miss. To address **3**, we implement an Agent which utilizes RAG and CoT prompting and grounds its generation in the Atomic Red Team repository¹. By retrieving valid command-line syntax from actual emulation tests, we prevent the agent from hallucinating invalid commands, ensuring the generated rules are valid and executable. Finally, to resolve **4**, we introduce an Infrastructure-Aware Filtering Agent. This agent ingests the organization's asset profile (as a JSON schema) and employs CoT reasoning to validate technique feasibility. It acts as a gatekeeper, ensuring that computational resources are only spent adapting rules that are theoretically detectable within the target environment.

4.3.2 Rule Ingestion

The initial phase of ARMS is the Rule Ingestion engine. Its primary objective is to normalize the heterogeneity of input detection rules into a structured, machine-readable format. This decoupling allows the downstream mutation engines to operate on abstract logic rather than parsing raw text repeatedly. The process is divided into two key components:

¹Atomic Red Team: <https://github.com/redcanaryco/atomic-red-team>

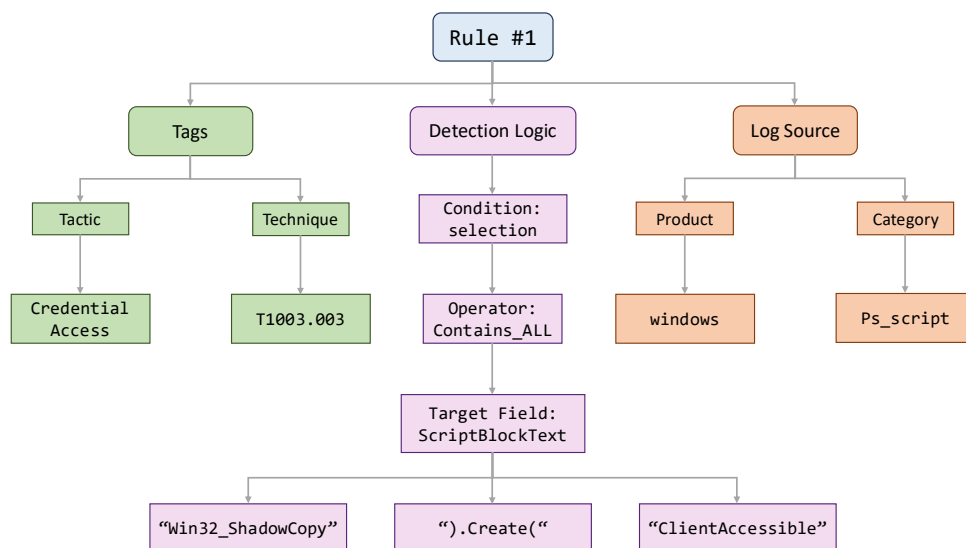


Figure 4.3: Illustrative example of a generated AST of a Sigma rule.

1. *Feature Extraction:* ARMS extracts the semantic metadata required to understand the adversarial context of the rule. It parses the input rule to identify critical indicators, including the tactics, techniques, and procedures mapped to the MITRE ATT&CK framework, as well as specific Indicators of Compromise such as file hashes, command-line arguments, or registry keys. This information is essential for the subsequent mutation step.
2. *Rule to AST Converter:* ARMS converts the rule into an Abstract Syntax Tree. While raw Sigma rules provide a structured format, they are still fundamentally text-based and rigid. Converting them into an AST provides a unified, hierarchical representation of the detection logic (*e.g.*, AND/OR conditions, selections, and filters) that is independent of the original syntax. Figure 4.3 illustrates an example of a generated AST structure of a Sigma rule from the SigmaHQ repository titled "Create Volume Shadow Copy with Powershell". As illustrated, this parsing process standardizes the rule into a unified programmatic structure. The resulting AST explicitly isolates the rule's MITRE ATT&CK tactics and techniques and the log source. More importantly, it captures the detection logic by extracting the target telemetry fields, their associated logical operators (*e.g.*, `Contains_ALL`), and the specific Indicators of Compromise, all while preserving the original structure of the selection field.

4.3.3 Rule Mutation

The core engine of ARMS is the Rule Mutation phase. This phase is responsible for transforming a static seed rule into a diverse set of behavioral variants. ARMS identifies alternative attack methods that achieve the same tactical goal as the input rule. It is critical to recognize that a detection rule does not merely represent an alert string; it represents a specific adversarial procedure. Therefore, the first task is to discover interchangeable techniques that can execute the same attack using different mechanisms. The rule mutation engine identifies variants by analyzing a technique’s location within a three-dimensional proximity space:

1. *Structural Proximity*: This dimension leverages the hierarchical taxonomy of the MITRE ATT&CK framework. We define techniques as structurally close if they minimize the shortest path within the MITRE ATT&CK Directed Acyclic Graph (DAG). To construct this DAG, ARMS parses the official MITRE ATT&CK knowledge. It extracts all `attack-pattern` objects to instantiate the graph nodes, and extracts `subtechnique-of` relationship objects to establish directed edges, thus logically linking each sub-technique to its overarching parent or other siblings. Specifically, techniques sharing the same parent technique are considered immediate siblings and highly probable variants. Figure 4.4a illustrates this process, where the seed technique T1059 (Command and Scripting Interpreter) is expanded into its sub-techniques (*e.g.*, T1059.001 PowerShell, T1059.003 Windows Command Shell), each representing a valid structural variant. However, if a technique does not have any parent or siblings, this proximity space will not find any variant techniques. This necessitates the need for other proximity spaces to find relevant techniques.
2. *Functional Proximity*: While structural proximity finds siblings, it misses techniques that are taxonomically distant but functionally similar. We define the functional proximity based on the semantic similarity of technique descriptions, relying on the hypothesis that techniques described with similar language likely serve the same adversarial intent. To find this set of techniques, ARMS performs Corpus Extraction [30, 72] by harvesting unstructured text descriptions and detection advice from the MITRE ATT&CK knowledge base. We then utilize an embedding model (*e.g.*, sentence-transformers model) to project these descriptions into a

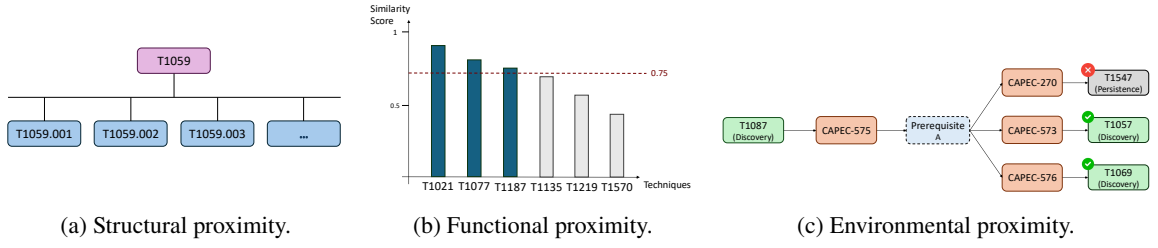


Figure 4.4: Example of three-dimensional variant discovery.

high-dimensional vector space. ARMS calculates the Cosine similarity between the original technique vector (V_o) and all candidate vectors (V_t), selecting variants that exceed a similarity threshold θ . $Sim(V_o, V_t) = \frac{V_o \cdot V_t}{\|V_o\| \|V_t\|} > \theta$. The threshold was set to 0.75. During preliminary tuning, we saw that thresholds lower than 0.70 resulted in noise and irrelevant techniques, while thresholds above 0.85 severely limited the discovery of valid functional variants. Figure 4.4b shows an example where ARMS identified T1021 and 2 others techniques similar to the original technique extracted from the rule.

3. *Environmental proximity*: It defines closeness based on the system state required to execute an attack. Two techniques are considered isomorphic if they rely on the exact same set of privileges, vulnerabilities, and resources, and they share the same tactic [59]. To find this set of techniques, ARMS employs cross-framework mapping [69] to link the seed’s MITRE ID to its corresponding CAPEC ID (Common Attack Pattern Enumeration and Classification), which contains richer dependency data. We parse fields such as Prerequisites to identify the specific environmental conditions required for the attack. ARMS then queries the CAPEC dataset to identify other attack patterns that share identical prerequisites. However, sharing a vulnerability does not automatically imply tactical relevance. To ensure the generated variants serve the same strategic goal as the original attack, we enforce a tactic consistency filter. ARMS only accepts techniques that align with the parent tactic of the seed rule, discarding those that rely on the same vulnerability but result in a completely different outcome. As shown in Figure 4.4c, ARMS starts with the seed technique T1087. It maps this to CAPEC-575 and extracts its specific prerequisites. Querying the knowledge base for other CAPEC IDs sharing these exact prerequisites yields CAPEC-270, CAPEC-573, and CAPEC-576. These patterns map back to three potential variant techniques: T1547, T1057, and T1069. Finally,

ARMS applies the tactic-based filtering. Since the original technique T1087 belongs to the Discovery tactic, the system discards T1547 as it belongs to the Persistence. T1057 and T1069 are retained as valid environmental variants because they satisfy the environmental dependencies while maintaining the original tactical intent.

4.3.4 Rule Optimization

The third step of ARMS is Rule Optimization. This phase functions as a strategic filter, positioned between the Rule Mutation and the Rule Adaptation components. Generating rules for techniques that are irrelevant to the organization's specific environment wastes computational resources during the adaptation phase and contributes to alert fatigue in the SOC. The goal of this step is to ensure that ARMS only produces rules that are tailored to the enterprise's active infrastructure.

```
1  {
2    "meta": { "enterprise_id": "ENTERPRISE-001", "security_level": "High" },
3    "global_constraints": { "allow_cloud": true, "allow_containers": true },
4    "active_platforms": ["Windows", "Linux", "Azure", "AWS", "Office 365"],
5    "platform_details": {
6      "Windows": {
7        "versions": ["Windows 11", "Server 2022"],
8        "roles": ["Domain Controller", "Workstation"]
9      },
10     "Linux": { "versions": ["Ubuntu 22.04 LTS"], "roles": ["Web Server"] }
11   },
12   "active_services": ["Active Directory", "Exchange", "Kubernetes", "Docker"],
13   "excluded_platforms": ["macOS", "Android", "iOS", "GCP", "..."]
14 }
```

Figure 4.5: Example of the Infrastructure JSON file.

1. *Infrastructure Analyzer*: This component ingests the raw infrastructure profile of the enterprise (e.g., ontology or graph-based representation). It performs preprocessing to normalize this data into a structured JSON format, creating a digital representation of the environment's capabilities and constraints (e.g., OS types, installed software, logging levels). Figure 4.5 provides an example of the JSON file.
2. *Filtering Agent*: The core of the optimization is an LLM-based filtering agent. Utilizing CoT reasoning, this agent evaluates each candidate variant technique against the infrastructure

profile. The agent determines applicability based on hard constraints. For example, if the infrastructure profile indicates a Windows-only environment with no Mac OS assets, any variant technique exclusive to Mac OS (*e.g.*, T1543.004 Launch Daemons) is immediately discarded. By filtering these out before the computationally expensive adaptation phase, ARMS ensures that resources are focused solely on generating high-value rules.

4.3.5 Rule Adaptation

After discovering the relevant technique variants, the final challenge is to adapt the detection rule’s syntax to match the new adversarial procedure. A direct syntax translation is often impossible because different techniques (*e.g.*, PowerShell vs. WMI vs. Bash) utilize fundamentally different command structures and logging artifacts. Naive heuristic methods (*if-else* logic) lack the scalability to handle the infinite variability of command-line obfuscation. Supervised learning approaches are equally infeasible due to the lack of large-scale, labeled datasets mapping rules to their variants. While zero-shot Generative AI [73] offers flexibility, it is prone to hallucination and lacks the grounded context required for security context. To address those challenges, ARMS employs a RAG Agent enhanced with CoT reasoning [68]. This hybrid approach grounds the LLM in verifiable ground truth data while leveraging its reasoning capabilities for syntactic adaptation.

1. *Grounding via Atomic Red Team:* To prevent hallucination, we utilize Atomic Red Team (ART) [74], a repository of scripted cyber attacks mapped to the MITRE ATT&CK framework. We treat the ART repository as our reference for translation. Since a single MITRE Technique ID (*e.g.*, T1003) is abstract and may have multiple distinct implementations (*e.g.*, using MimiKatz, ProcDump, or TaskMgr), we cannot simply retrieve any or all tests.
2. *Semantic Alignment Algorithm:* To adapt the rule accurately, ARMS identifies the specific implementation of the target technique that aligns with the original rule’s intent. The selection process is as follows: (i) fetch all available Atomic tests associated with the target variant technique ID; (ii) encode the original rule’s detection logic and description into a high-dimensional vector using an embedding model; (iii) perform Cosine similarity against the vectorized descriptions of the retrieved Atomic tests; and (iv) identify the single Target

test that is semantically closest to the original rule. This ensures we are adapting the rule for the correct atomic test of the new technique.

3. *Context Injection:* Once the target test is selected, we construct a precise translation context containing two ground truths: (i) the specific Atomic test command that triggers the original rule (*i.e.*, source context), and (ii) the specific Atomic test command for the new variant selected in the target context.
4. *Generative Execution:* We feed the translation pair to the LLM agent. We instruct the agent to analyze the syntax of the source command, map the underlying detection logic to the target command structure, and craft the Sigma rule entirely. This ensures the new rule detects the behavior of the variant, not just keywords. Figure 4.6 shows the prompt setup for the rule

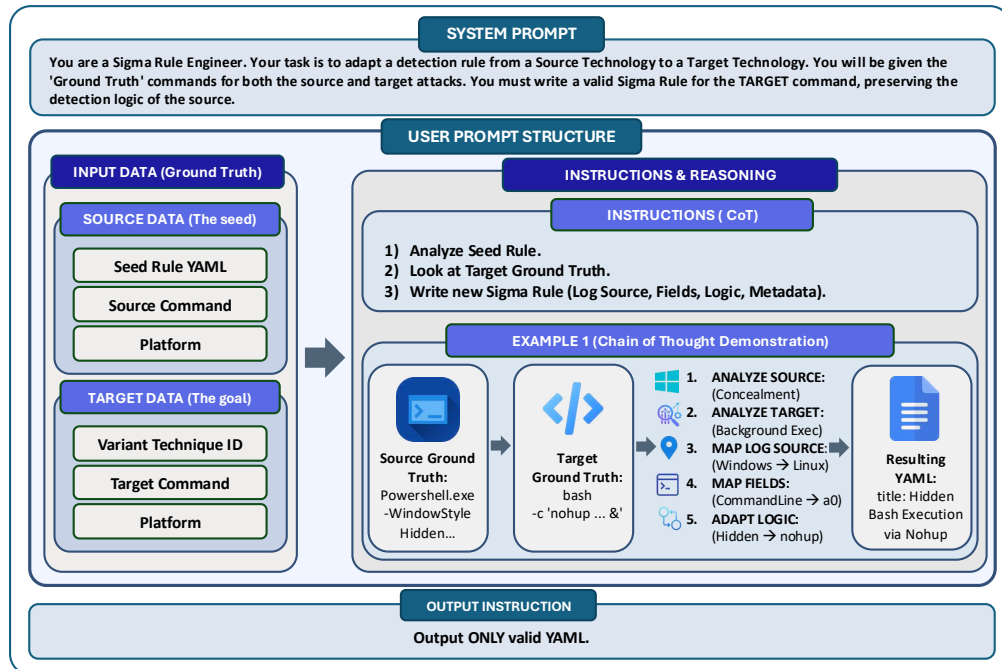


Figure 4.6: Overview of the prompt given to the RAG-based Adaptor Agent.

adaptation phase. Initially, there is a system prompt that clearly specifies the task the model is expected to perform, which helps prevent hallucinations. Next, the user prompt section is presented, containing the ground truth data for both the original and target techniques derived from ART data. This section also includes instructions and reasoning steps. The instructions

break down the task into several steps, and through a few-shot learning approach, an example of the CoT reasoning is provided. This ensures the model understands the steps to follow and can iterate through a complete generative process. Finally, the output instruction specifies the desired format for the model’s final output. This comprehensive process is designed to ensure the model delivers the best possible results.

5. *Rule Validator*: This is a deterministic module that uses public packages/tools to parse the generated rule to ensure compliance with the Sigma schema, automatically fixing common formatting errors before saving the file for deployment.

4.4 Implementation

We build a Proof of Concept of ARMS using Python 3.10 (see Appendix F for PoC). The experiments were conducted on Ubuntu 20.04 with a 64-core processor, 211 GB RAM, and an NVIDIA A100 GPU.

4.4.1 LLM Models

There are generally two methods to work with LLMs: (i) using a cloud-based LLM or (ii) using a locally hosted LLM. Cloud-based LLM access does not require extensive computational resources and simplifies request handling; however, it is often associated with usage costs, and concerns remain about data confidentiality and the possible influence of concurrent API requests on output quality. Running LLMs locally ensures data privacy and eliminates risks associated with multi-tenant environments, but demands significant computing resources. Without sufficient hardware, one must use lighter LLM variants, which may negatively impact performance. To evaluate ARMS, we used (i) *Foundation Sec* [75] a fine-tuned version of Meta’s Llama 3.1 model for cybersecurity; (ii) *Lily Cybersecurity* [76] a Mistral fine-tune model with 22000 hand-crafted cybersecurity and hacking-related data pairs; and (iii) *ChatGPT-4o-mini* [77] through OpenAI APIs. We set the temperature parameter to 0.1 and the maximum token length to 2000 to maintain consistency in output quality and length. In addition to that, we used *all-mpnet-base-v2* [78] for functional proximity. It is a sentence-transformers model that maps sentences and paragraphs to

a 768 dimensional dense vector space and can be used for tasks like clustering or semantic search.

4.4.2 Public Sigma Rules Repository

We constructed a seed dataset by randomly selecting 50 verified detection rules from the official SigmaHQ repository [48]. To ensure compatibility with ARMS’s multi-stage pipeline, we made sure that the MITRE technique associated with the selected input rule must have at least one corresponding test case available in the Atomic Red Team library.

4.4.3 Knowledge Base

ARMS incorporates three distinct external knowledge bases to drive its reasoning and mutation engines: (i) MITRE ATT&CK knowledge base [43], which provides the taxonomy required for our structural and functional variant discovery. We utilized the MITRE ATT&CK Enterprise knowledge base version 14.1, which encompasses 14 tactics, 201 techniques, and 435 sub-techniques; (ii) CAPEC knowledge base [69] version 3.9, which contains 559 distinct attack patterns. CAPEC provides the rich prerequisite data necessary for the environmental proximity analysis; and (iii) Atomic Red Team repository [74] that provides concrete, executable command-line tests. The repository contains over 1,300 executable atomic tests that cover more than 300 unique MITRE ATT&CK techniques.

4.5 Evaluation

Unlike standard detection engineering tasks [28], there is no pre-existing dataset/baseline that maps rules to their optimal, infrastructure-aware variants. To evaluate ARMS, we address four research questions (RQs) designed to assess ARMS’s output from complementary perspectives.

RQ1	Fidelity Verification	Are the generated outputs true behavioral variants of the original attack?
------------	----------------------------------	--

This RQ aims to verify if ARMS’s outputs represent the same underlying adversarial intent rather than unrelated noise or hallucinations. Measuring the fidelity of security logic is inherently

challenging; traditional similarity metrics (such as BLEU [79] or cosine similarity) are insufficient because they cannot discern whether a syntactic change preserves the rule’s detection logic or renders it invalid. Consequently, relying solely on automated similarity scores yields unreliable results. To address this, we applied a cross-validation by: (i) asking a security practitioner to evaluate the validity of the generated rules; and (ii) employing an LLM-as-a-Judge: the first LLM analyzes the original seed rule to extract the core adversarial intent, while the second LLM acts as a judge and determines if the variant preserves the original intent while correctly adapting to the new environment.

Table 4.1: Correctness of rules generated by ARMS compared with baselines.

Metric	ARMS Foundation Sec	ARMS Lily	ARMS ChatGPT	ChatGPT-off-the-shelf
Total	173	179	168	93
Valid	166 (95.9%)	159 (88.8%)	154 (91.7%)	51 (54.8%)
Invalid	7 (4.1%)	20 (11.2%)	14 (8.3%)	10 (10.8%)
Unchanged	0 (0.0%)	0 (0.0%)	0 (0.0%)	32 (34.4%)

Table 4.1 presents the comparison results between ARMS and the other baseline models. Here, we measured three metrics: (i) valid variant: the rule has been successfully mutated and correctly adapted to the new technique/platform while preserving the original intent; (ii) invalid variant: the mutation resulted in broken logic, or a loss of the original detection capability; (iii) unchanged variant: the system failed to trigger a mutation, resulting in a duplicate of the input rule. We can see that ARMS generated a higher number of total variants (173, 179, 168 vs. 93). This is because of the rule mutation component that actively explores for new techniques. In terms of quality, ARMS demonstrates superior fidelity. ARMS achieves a fidelity rate of 95.9% (166 out of 173 variants), 88.8% (159 out of 179 variants), and 91.7% (154 out of 168 variants), respectively under different configurations, compared to only 54.8% (51 out of 93) for the baseline. This significant margin validates the effectiveness of the rule mutation component in maintaining semantic correctness during mutation. Notably, ARMS achieved an unchanged rate of 0.0%, whereas the baseline resulted in 34.4% unchanged outputs. This indicates that ChatGPT-off-the-shelf often struggles to escape the local optima of the original rule, frequently returning the input itself. In contrast, ARMS consistently enforces mutation, ensuring that every output was a distinct attempt at adapting the logic to a new technique or platform.

RQ2	Rule Correctness	Are the generated rules semantically valid and convertible to SIEM queries?
-----	-------------------------	---

To assess the operational viability of the generated rule, we decompose the concept of rule correctness into two distinct dimensions: syntactic validity and semantic logic.

Table 4.2: Correctness of rules generated by ARMS compared with baselines.

Baseline	Syntactic and Convertibility	Condition Accuracy
ARMS Foundation Sec	173 / 173 (100.0%)	173 / 173 (100.0%)
ARMS Lily	171 / 179 (95.5%)	169 / 179 (94.4%)
ARMS ChatGPT	168 / 168 (100.0%)	166 / 168 (98.8%)
ChatGPT-off-the-shelf	93 / 93 (100.0%)	90 / 93 (96.8%)

1. *Syntactic Validity and Convertibility:* A rule that fails to convert into a target SIEM format is a failure, regardless of its theoretical quality. To evaluate this, we define the rule convertibility as the percentage of generated Sigma rules that can be successfully converted into Splunk and ELK queries without triggering parsing errors or syntax warnings. We utilized the official `sigma-cli` [49] tool to attempt this conversion for every generated variant. Table 4.2 shows that ARMS achieves a rule convertibility rate of 95.5% while using Lily LLM and 100% when using Foundation sec and OpenAI LLM, successfully converting the generated variants into valid SIEM queries. The perfect conversion score of ARMS can be attributed to the structured enforcement of the Sigma schema within the Rule adaptation phase, which first uses CoT reasoning to construct the rule, and second, ensures that all outputs strictly adhere to the required YAML format before they are finalized.
2. *Semantic Logic and Condition Accuracy* Beyond simple compilation, the internal logic of the rule must be operationally sound. A rule might be syntactically perfect but operationally useless if it contains broken boolean logic, such as impossible AND conditions or disconnected selection fields. To measure this, we adopt the condition accuracy metric established in [20]. This metric specifically evaluates the condition field of the Sigma rule to determine if the boolean operators (AND, OR, NOT) correctly connect the detection artifacts to form a coherent

query. Table 4.2 shows that ARMS also achieves a condition accuracy of 94.4% minimum, indicating that most generated variants possessed logically coherent conditions, except in rare cases when the description is too vague or an incomplete rule has been generated.

RQ3	Ablation Study	What is the effect of different components of ARMS?
-----	-----------------------	---

To evaluate the contribution of individual components within the ARMS architecture, we conducted an ablation study focusing on two critical phases: the three-dimensional proximity space and the rule optimization process. This analysis aims to quantify the impact of each proximity space and the optimization module on the total volume, fidelity, and relevancy of the generated rules.

1. *Impact of three-dimensional proximity space:* We compared the full ARMS pipeline against configurations where the structural, functional, or environmental proximity spaces were individually removed. The goal was to determine how each knowledge graph component contributes to the generation of valid attack variants. As presented in Figure 4.7, the removal of any single proximity space results in a measurable decrease in the total number of generated variants, confirming that each component actively expands the search space for potential threats. The most significant impact is observed when the structural proximity space is removed; the total number of variants drops by exactly half (from 168 to 84). This indicates that structural analysis is the primary driver for identifying technique variants, likely due to its direct mapping of code-level relationships. The functional proximity space follows as the second most influential component, while the environmental proximity space shows a slightly smaller but still distinct reduction in output. The lesser impact of the environmental component can be attributed to the current limitations of the CAPEC dataset, which does not yet fully cover every possible technique. However, as these external knowledge bases expand, we anticipate the influence of this component to grow. The fidelity percentages remains high and relatively stable across all configurations.
2. *Impact of Rule Optimization:* We compared the full ARMS pipeline against a version where the LLM-based filtering mechanism was disabled. Figure 4.8 highlights the critical role of

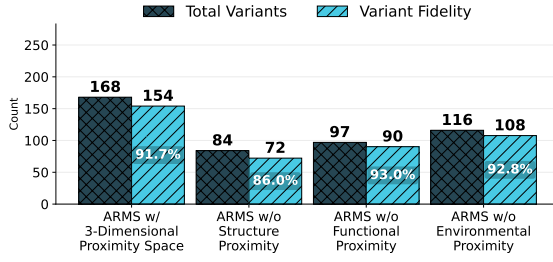


Figure 4.7: Impact of three-dimensional proximity space on the rule fidelity.

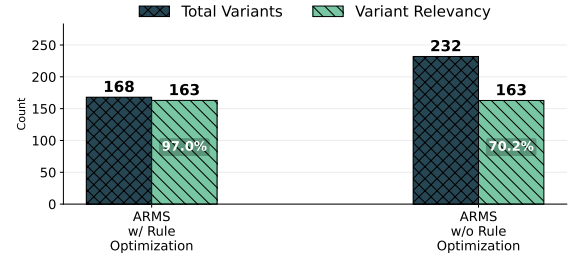


Figure 4.8: Impact of rule optimization on rule relevancy.

the optimization phase in aligning generated rules with the target infrastructure. Without the filtering component, the system generated a higher total volume of 232 variants, but the infrastructure relevancy dropped significantly to 70.2%. This indicates that nearly 30% of the generated rules were irrelevant to the specific environment, a scenario that would directly contribute to alert fatigue for security operations teams. In contrast, the ARMS with rule optimization, which incorporates the filtering mechanism, achieved a 97.0% relevancy rate (163 out of 168 rules). This demonstrates that the optimization phase successfully prunes incompatible or redundant techniques, ensuring that the final output remains highly targeted and actionable for the deployed infrastructure.

4.6 Discussion

The assessment of ARMS underscores its capability to shift detection engineering from a reactive, manual process to a proactive, automated field. A key insight from our study is the notable difference in generative behavior between our structured solution and typical large language models. We observed that off-the-shelf models, have a strong tendency towards *identity mapping*. They often reproduce the original rule instead of creating a valid variants. This phenomenon suggests that without external guidance, general and off-the-shelf models fail to escape the local optima of the input. By forcing the generation process through our three-dimensional proximity space, ARMS effectively breaks this cycle, ensuring that every generated rule represents a distinct, meaningful mutation of the adversarial behavior rather than a mere reproduction. We also observed the performance of ARMS under different LLM configurations. The results show the solid performance of ARMS regardless of the underlying model, as long as the generation process is grounded with the

proposed pipeline. Consequently, rather than prescribing a universally optimal LLM, the choice of model should be dictated by the specific operational constraints and priorities (*e.g.*, inference speed, deployment costs, or data privacy requirements) of the deploying enterprise. While the multi-layered discovery approach has proved to be essential for uncovering non-obvious, semantically equivalent threats, ARMS's effectiveness is inherently limited by the completeness of the external knowledge bases it utilizes. The reliance on CAPEC for environmental prerequisites and Atomic Red Team for rule adaptation creates a dependency constraint; if a technique lacks a corresponding atomic test or a structured prerequisite definition, ARMS cannot reliably generate or validate a variant.

Chapter 5

Discussion: Toward a Proactive APT Detection Lifecycle

This chapter connects the two main contributions of the thesis: **AUTOSigma** (Chapter 3) and **ARMS** (Chapter 4). Rather than treating them as two separate systems, we position them as complementary stages in a single detection lifecycle for APT hunting. In this perspective, **AUTOSigma** provides the entry point of the lifecycle by operationalizing intelligence, while **ARMS** extends that initial detection logic to maintain coverage as adversaries modify their behavior.

5.1 What **AUTOSigma** and **ARMS** each contribute

AUTOSigma: rapid CTI operationalization: **AUTOSigma** targets the CTI-to-detection bottleneck by transforming unstructured threat intelligence into Sigma rule candidates [3,4,48]. In practice, **AUTOSigma** treats CTI reports as the starting point for detection engineering and produces a structured set of seed rules. As shown in Chapter 3, this is achieved through contextual analysis of the report, enrichment when key operational details are missing, decomposition of attack scenarios into smaller attack steps, rule grounding from verified sigma repositories, and LLM-as-a-judge refinement.

AUTOSigma limitations: **AUTOSigma**'s effectiveness depends on the CTI it consumes and the observability assumptions in that report. CTI reports are written for analysts, not for execution, and they often leave out the telemetry details needed to construct precise rule conditions. In addition,

Table 5.1: How AUTOSigma and ARMS complement each other as stages in a unified detection engineering lifecycle.

Aspect	AUTOSigma	ARMS
Input	CTI report text [3,4]	Seed Sigma rule + infrastructure profile (asset and telemetry constraints)
Output	Seed Sigma rule set for hunting and monitoring	Variant rule set filtered for feasibility and operational relevance
Main objective	Turn narrative CTI into deployable Sigma rules [48]	Expand and adapt seed rules into resilient, infrastructure-aware rule variants [43]
Main strength	Speed of operationalization from narrative intelligence	Sustained coverage under technique substitution and environment shifts

the resulting rules are typically anchored to a specific instance of an attack. This is a useful starting point for threat hunting, but it remains fragile when adversaries change techniques, tools, platforms, or execution primitives to achieve the same objective [4].

ARMS: sustaining coverage under technique substitution: ARMS addresses the operational reality that even a correct rule can become brittle once an adversary changes the way they execute the same tactic. This includes tool substitution, technique switching, and cross-platform shifts that move activity outside the detection scope of the original rule [26]. Starting from a seed rule, ARMS expands coverage by generating infrastructure-aware rule variants guided by the three proximity spaces for technique discovery and by grounded generation utilizing RAGs and CoT reasoning [51, 70]. This positions ARMS as a maintenance and resilience component.

ARMS limitations: While Rule variants expand the detection coverage, they can also increase the size of the rule set; without filtering and prioritization, variant generation can shift effort from writing rules to curating them. This concern is closely related to the larger operational burden of maintaining large rule collections [25].

5.2 How the two systems connect

Artifacts and data flow: The integration point between AUTOΣIGMA and ARMS is the detection rule artifact itself. AUTOΣIGMA produces a set of Sigma rules that encode detection rules in a platform-agnostic form [48]. These rules become inputs to ARMS, which generates and filters variants so that coverage persists under technique substitution and remains compatible with the infrastructure.

Figure 5.1 illustrates the overall workflow at a system level. Operationally, the combined process can be described in three main steps:

1. *Seed generation:* CTI text is ingested and interpreted by AUTOΣIGMA, producing a set of Sigma seed rules aligned to the observed behaviors in the report [3,4].
2. *Variant expansion:* Seed rules become inputs to ARMS, which identifies plausible technique substitutions using MITRE ATT&CK as a behavioral index and generates the corresponding rule variants [43].
3. *Environment alignment:* Variants are filtered and selected for deployment based on infrastructure and available telemetry, reducing the likelihood of deploying inapplicable rules [27,28].

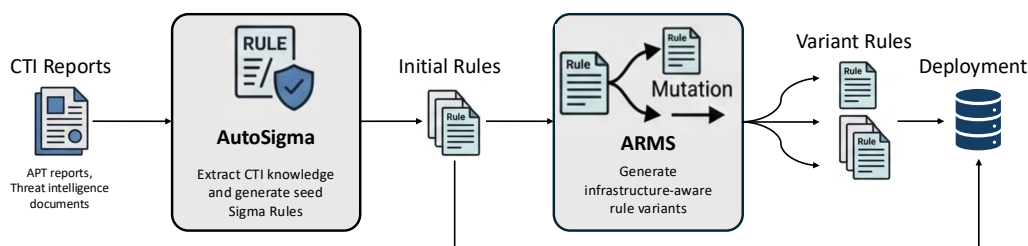


Figure 5.1: Pipeline view of the unified workflow: AUTOΣIGMA produces Sigma seed rules from narrative CTI, and ARMS expands these seeds into infrastructure-aware variant sets suitable for deployment.

5.3 Unified detection engineering lifecycle

Threat hunting workflows are inherently iterative. Intelligence informs detection changes, detections produce alerts, and outcomes feed into refinement decisions [1]. The contribution of this thesis is to make that loop concrete for rule-based detection by defining a lifecycle with explicit stages

and artifacts. In this lifecycle, AUTOSIGMA supports intelligence operationalization, while ARMS supports resilience under variation.

The lifecycle can be summarized as follows:

1. *CTI acquisition*: Collect narrative intelligence from threat reports and incident write-ups [3, 4].
2. *Detection synthesis*: Produce seed Sigma rules to capture behaviors described in CTI in a reusable representation [48].
3. *Behavioral indexing*: Associate detections with relevant MITRE ATT&CK tactics and techniques to support reasoning about coverage and substitutions [43].
4. *Variant generation*: Expand coverage by generating behaviorally related rule variants, guided by retrieval and structured reasoning [51, 70].
5. *Infrastructure filtering*: Remove or adapt variants that require unavailable telemetry, platforms, or schema support in the target environment [27, 28].
6. *Deployment and iteration*: Deploy selected rules through standard SOC processes and refine based on alert outcomes and operational feedback [1].

This framing also clarifies the division of labor between automation and human oversight. Automation accelerates rule synthesis and variant exploration, while analysts remain responsible for policy decisions, tuning, and validation under operational constraints.

5.4 Limitations

Deployment realism: A practical deployment requires acknowledging that Sigma rules represent detection intent rather than a single universal execution format. Organizations typically convert and tune rules to match local schemas and SIEM backends, then iterate based on observed alert quality [28, 48]. In this context, the combined AUTOSIGMA + ARMS workflow is naturally aligned to staged deployment: seed rules can be reviewed and validated first, then expanded into variants once the initial detection intent is confirmed.

False positives and rule set growth: Variant generation can widen coverage, but it also risks noise. This is where infrastructure filtering and rule set management matter. If variant generation produces

multiple overlapping detections, SOC teams need prioritization and consolidation strategies to avoid alert fatigue [25]. In practice, the most effective operational posture is to treat variants as candidates, not defaults, and deploy them incrementally with clear acceptance criteria.

Model and generation risks: Both systems rely on LLMs, which introduces the risk of producing hallucinations. This is a known issue in LLM-assisted generation and evaluation settings [56, 73]. The mitigation strategy in this thesis is to constrain generation through grounding, reasoning, and validation, but the operational takeaway remains that automated outputs should enter the SOC workflow as reviewable artifacts.

Chapter 6

Conclusion

6.1 Conclusion

This thesis tackled a key problem in threat hunting: the systematic transformation of high-level cyber threat intelligence into effective, adaptable detection capabilities for evolving Advanced Persistent Threat (APT) campaigns. We began by identifying an operational gap. Narrative CTI reports contain rich descriptions of adversary behavior, but converting those narratives into actionable detection rules is labor-intensive and depends on expert analysts. At the same time, once deployed, detection rules remain static even as the threat landscape changes. Attackers often swap tools, alter techniques, or exploit new platforms to evade fixed defenses. Bridging the gap between intelligence and detection, while maintaining effective coverage with adversarial evolution, was the main motivation for this work.

To address the first part of this challenge, we introduced **AUTOSigma**, a solution that converts narrative CTI into structured Sigma rules. **AUTOSigma** applies contextual analysis and knowledge enrichment to the report text, decomposes multi-stage attack narratives into attack steps, uses existing rule templates for grounding, and refines rules with an LLM-as-a-judge feedback loop. Our evaluation showed that **AUTOSigma** can generate high-quality Sigma rules aligned with source intelligence, while reducing the manual effort required from security practitioners. However, generating rules from CTI alone is insufficient when adversaries can adapt. A rule derived from one report captures only that specific instance of attacker behavior. In practice, APT groups frequently alter how they

achieve the same goal. A rule for one technique will miss an alternative technique that accomplishes the same objective. To address this, we developed ARMS, an automated rule mutation solution. ARMS treats each seed Sigma rule as a starting point and systematically generates behaviorally equivalent rule variants. It achieves this by searching for related techniques in the MITRE ATT&CK knowledge base and using retrieval-augmented generation (RAG) with chain-of-thought reasoning to generate variant detection rules. In this manner, ARMS transforms each static detection rule into an evolving set of variants, ensuring that detection coverage persists when attackers change tools or target different environments. Another key contribution of this thesis is the unified detection engineering lifecycle that integrates these capabilities. AutoSigma and ARMS complement each other: AutoSigma accelerates the creation of initial detection content from CTI, while ARMS sustains and expands that content in the face of change. A practical workflow might follow steps like: 1. *CTI interpretation*: Extract attack behaviors from threat intelligence reports. 2. *Rule synthesis*: Generate seed Sigma rules (AutoSigma) that capture those behaviors. 3. *Behavioral indexing*: Link rules to MITRE ATT&CK tactics and techniques to enable variant reasoning. 4. *Rule mutation*: Generate behaviorally related rule variants (ARMS) to cover technique substitutions. 5. *Deployment and monitoring*: Deploy selected rules to security monitoring systems, monitor alerts, and feed back the output into the process. This cycle transforms detection engineering into a continuous process. It supports rapid response to new intelligence and allows defenses to evolve alongside adversarial behavior.

Despite these contributions, several limitations remain. Both AutoSigma and ARMS rely on external knowledge sources such as MITRE ATT&CK, Sigma rule repositories, and related threat intelligence resources. As a result, the quality and completeness of these knowledge bases influence the effectiveness of the generated rules and variants. In addition, because both systems utilize large language models for certain stages of generation and reasoning, the possibility of model hallucination or incorrect inference remains a concern that needs careful validation.

6.2 Future Research Directions

This work opens several avenues for future research. One promising direction is to integrate live telemetry feedback into the loop, so that real alert outcomes (for example, false positive rates or missed detections) guide automated rule refinement and prioritization. Another area is automated rule consolidation and optimization. As ARMS can generate many variants, methods to detect redundancy or merge overlapping rules would help manage the overall rule set size. Developing formal metrics of semantic equivalence between rules could also be valuable. Such metrics would let us rigorously assess how close a variant is to the original intent. Incorporating adversarial testing frameworks is another direction: by simulating an attacker’s evasion strategies, we could test generated rules and identify weak spots. Finally, extending to more detection languages or platforms (beyond Sigma) would improve interoperability in diverse enterprise environments. In conclusion, this thesis demonstrates that careful automation can significantly speed up both the initial creation of detection logic and its ongoing adaptation. Human analysts remain essential for policy decisions and validations, but tools like AUTO SIGMA and ARMS can dramatically reduce their routine burden. By transforming threat intelligence into dynamic, evolving detection content, we move toward a more proactive and resilient defense posture in the face of evolving APT threats.

Appendix A

LLM Prompting for AUTOSigma

Basic prompts such as "extract NERs from a given text" proved inadequate for cybersecurity tasks, which demand higher specificity and structure. To address this, we employed prompt engineering techniques that included explicit task instructions, relevant examples, and clearly defined output formats. An illustration of our structured prompt engineering is shown in Fig. A.1.

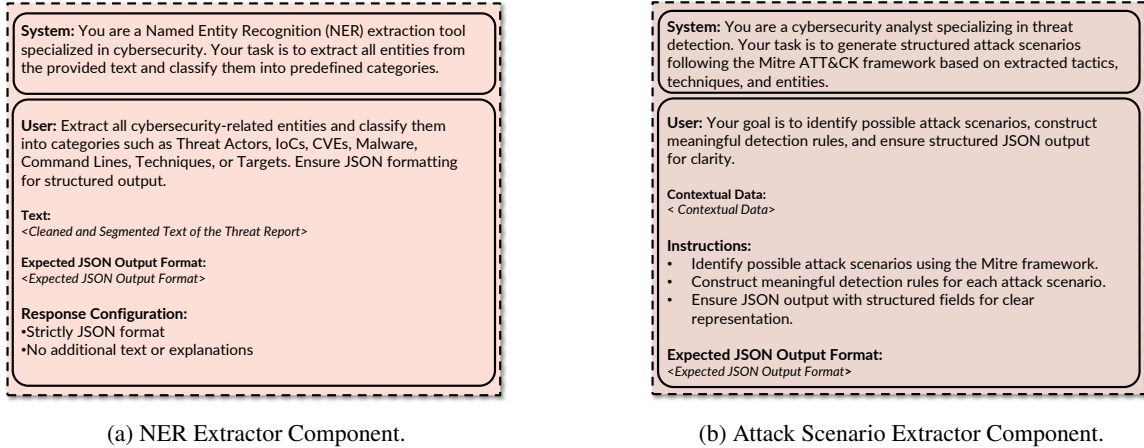


Figure A.1: Example of prompts used by AUTOSigma.

LLM-as-a-Judge Loop: To ensure the quality of the generated Sigma rules, AUTOSigma employs an LLM-as-a-Judge loop in which a generator and a validator model work iteratively. The generator produces an initial Sigma rule (as shown in Fig. A.2a), a separate LLM—used as the validator—then evaluates the generated rule against predefined criteria, including correctness, completeness, and Sigma syntax compliance (Fig. A.2b). If the validator’s score falls below a fixed threshold (8/10

System: You are a cybersecurity expert. Your task is to enrich a given Sigma rule by incorporating attack-specific details while ensuring it remains in valid YAML format.

User: Your goal is to modify and refine the Sigma rule to accurately detect the new attack scenario, incorporating previous feedback where available.

Input Data:

- Original Sigma Rule:**
<Closest Sigma Rule, Discovered in the previous component>
- New Attack Scenario:**
<attack description>
- Previous Feedback:**
<previous feedback> (if applicable)

Instructions:

- Ensure the Sigma rule is enhanced to match the attack scenario.
- Remove any Markdown artifacts (like triple backticks).
- Maintain proper YAML formatting and correctness.

Expected Output Format:
< Expected Output Format >

(a) Prompt used for Sigma rule generator.

System: You are an expert in Sigma rule validation. Your task is to assess the correctness and effectiveness of a Sigma rule generated for a specific attack scenario.

User: Evaluate the generated Sigma rule based on the attack description and rank its quality, providing structured feedback.

Input Data:

- Attack Description:**
<attack description>
- Generated Sigma Rule:**
<generated sigma rule>

Instructions:

- Verify if the rule accurately detects the attack scenario.
- Ensure correct log source alignment.
- Check if the detection logic is valid and practical.
- Identify any potential false positives.
- Validate proper YAML formatting.

Expected Output Format:

- Ranking:** Score the rule from 1-10.
- Feedback:** Provide suggestions for improvements.
- Corrections (if needed):** Highlight necessary modifications.

(b) Prompt used for Sigma rule validator.

Figure A.2: LLM-as-a-Judge prompts.

across all dimensions), its feedback is returned to the generator, and the refinement loop continues until the quality criteria are met. Furthermore, as illustrated in Fig. A.2, the prompts used for LLM-as-a-judge is adaptive and dynamically constructed.

Appendix B

Proof of Concept (PoC) (AUTOSigma)

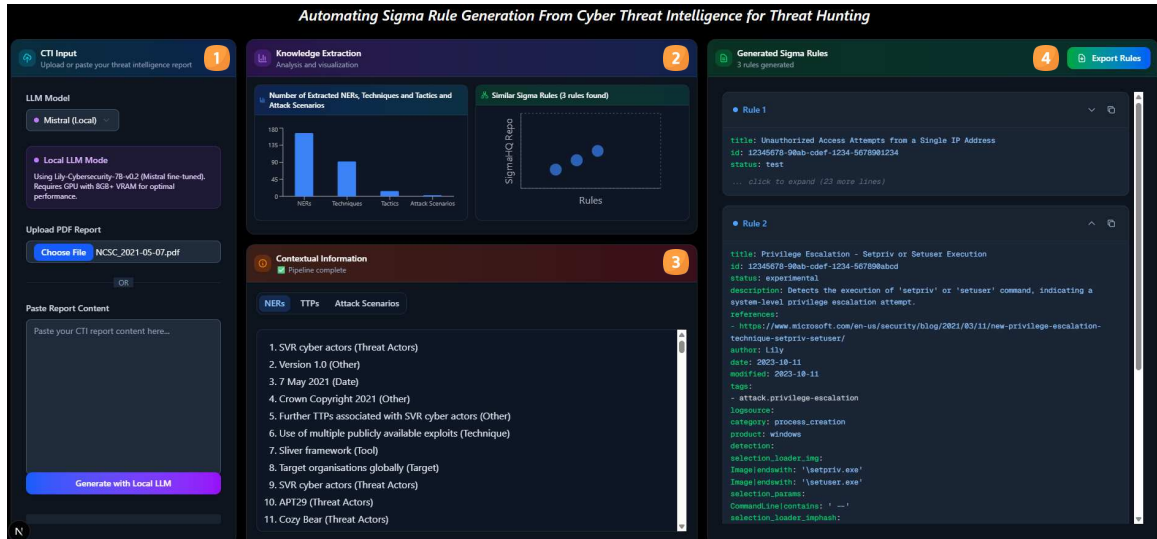


Figure B.1: Screenshot of the AUTOSigma PoC interface.

Fig. B.1 presents the Proof of Concept interface we developed for AUTOSigma, highlighting the end-to-end workflow of the AUTOSigma. The UI is structured into four main sections, visually indicated by blue rounded-square icons:

- 1 Input Area:** Users select the LLM model (local or cloud-based) and upload or paste the threat report to initiate analysis.
- 2 Knowledge Extraction & Similar Rule Discovery:** The left panel visualizes key statistics such as the total number of extracted Named Entities (NERs), MITRE ATT&CK techniques,

tactics, and attack tests. The right panel displays a bubble chart summarizing similar Sigma rules identified from the SigmaHQ repository.

3 Contextual Information: This area shows the detailed breakdown of contextual information extracted by `AUTOSigma`, including recognized NERs, ATT&CK techniques, tactics, and the reconstructed attack scenario.

4 Outputs and Rules: The final output section displays the generated Sigma detection rules and allows users to review or export them.

Appendix C

CTI Reports Summary

Table [C.1](#) provides a summary of the used CTI reports per APT in our evaluation. To reflect the diversity and realism of operational CTI, we selected reports from multiple security vendors, written at different levels of technical depth, and varying significantly in both length and structure. These reports mix multiple forms of information, including high-level executive summaries, detailed technical analyses, command line snippets, screenshots, figures, and narrative descriptions of attacker behavior. Using such a varied corpus demonstrates that AutoSigma can operate effectively across heterogeneous and unstructured report formats, regardless of whether the content is tactical, technical, or strategic.

Table C.1: Summary of CTI reports used in the evaluation.

	Report	Target Industries	Main TTPs	Published	Pages
APT41	Mandiant#1 [3]	Banking, finance, government, high-tech	PowerShell usage, Cobalt deployment, FTP for payload delivery	2020-03-25	13
	Mandiant#2 [80]	Gaming, healthcare, high-tech	Vulnerability exploitation, web shells, backdoors	2019-08-19	9
	Mandiant#3 [81]	Telecoms, news/media, software firms	Backdoors, reconnaissance, credential theft	2019-08-07	68
	Mandiant#4 [82]	U.S. state government, insurance, telecom	SQL injection, zero-day exploits, DNS manipulation	2022-03-08	9
	TrendMicro [83]	Enterprises, government, airline, publishing	StealthVector, backdoors, DNS manipulation	2021-08-24	55
APT28	ANSSI [84]	Government, universities, think tanks (France)	CVE-2023-23397, phishing, compromised edge devices	2023-10-26	7
	ClearSky [85]	Media, public information outlets (US, EU, Israel)	Disinformation, fake news, infrastructure reuse	2024-02-22	29
	Insikt#1 [86]	Ukrainian government	Roundcube/Outlook phishing (CVE-2023-23397), credential theft	2023-06-20	17
	Logpoint [87]	NATO-aligned government, defense, media	GooseEgg exploit, Print Spooler abuse, LOLBINs	2024-04-05	29
	Maverits [88]	Government, foreign affairs (Ukraine, Europe)	Backdoors, LOLBINs, cybercrime collaboration	2023-01-27	32
APT29	CrowdStrike [89]	Foreign ministries, NATO governments	Credential phishing, malware loader	2022-01-27	21
	HKUK [90]	Multiple UK sectors including academia	Password spraying, VPN compromise, supply chain	2023-11-14	12
	Insikt [91]	Think tanks, foreign policy institutions	Credential theft via phishing themes	2023-07-27	19
	Mandiant#5 [92]	U.S. and European government organizations	Credential theft, C2 tunneling, phishing kits	2018-11-19	17
	NCSC [93]	Defense, law, government entities	Credential spraying, Exchange exploitation	2021-05-07	7

Appendix D

Local LLMs Versus Cloud LLMs

While our evaluation compared both cloud-based and local LLM deployments, it is important to reiterate that **AUTO**SIGMA was designed as a pipeline, not as a benchmark for individual LLMs. The performance gaps we observed between the two settings reflect differences in model capabilities at specific stages of the pipeline rather than shortcomings in the design. In fact, our results suggest that each class of model brings unique strengths to different parts of the system. The local LLM used in our experiments—a fine-tuned variant of Mistral with a cybersecurity-specific training corpus—performed particularly well during the early analytical stages. Tasks such as NER and TTP extraction, and attack scenario extraction benefited from the model’s domain alignment. In these steps, the local model demonstrated strong precision and contextual understanding, likely due to the security-specific patterns it had learned during fine-tuning. However, the final stages of the pipeline—especially the generation of Sigma rules—placed greater demands on generative capacity, instruction adherence, and formatting consistency. Here, the cloud-based models exhibited a clear advantage. Backed by significantly larger parameter counts and broader pretraining data, these cloud-based models consistently produced more well-formed, coherent, and complete Sigma rules. Because our validation and scoring processes take place after generation, the strength of the initial drafts plays a decisive role in final outcomes. This explains why, even though the local model handled upstream analysis well, the API models slightly outperformed it in downstream metrics like rule validity and semantic alignment. This analysis also clarifies that the observed performance delta is not due to the pipeline failing on local models, but to generative limitations that the generator and

validator alone cannot fully overcome. It also opens a path for hybrid approaches—such as using a local model for extraction and an API model for generation—offering future work opportunities to balance performance and deployment constraints more effectively. It is also worth noting that, as discussed earlier, cloud-based LLMs introduce a small operational cost, while removing the need for computational resources. In our experiments, the cloud-based versions of AUTO-SIGMA had an estimated cost of \$0.0015 - \$0.003 per run, depending on the number of iterations the LLM-as-a-judge loop takes. Since these models require no dedicated GPU resources or local model hosting, they make them convenient for lightweight deployment. Conversely, local LLMs are free to use, yet demand sufficient hardware capacity, typically GPUs, to deploy. These trade-offs highlight an important design consideration for practitioners.

Appendix E

Average Evaluation Cross Multiple APTs

As shown in Table E.1, the trend observed for APT41 holds consistently for APT28 and APT29 as well. AUTO SIGMA significantly improves IoC coverage using both cloud-based and local LLMs. With cloud-based LLMs, the average IoC coverage increases from 20.1% (ChatGPT-off-the-shelf baseline) to 90.4%. Similarly, in the local LLM setting, AUTO SIGMA improves the average IoC coverage from 20.1% to 60.7%, corresponding to an increase of 40.6 percentage points. This observation is also consistent with our MITRE ATT&CK coverage results (see Table 3.4 in the main text), demonstrating that AUTO SIGMA not only matches the report-derived technique coverage, but also significantly broadens it, yielding a more comprehensive Sigma rule set that covers more techniques across different APT campaigns. The same pattern appears when considering semantic relevancy. For complex reports in APT28 and APT29, the average relevancy of AUTO SIGMA is high (*i.e.*, 0.816 and 0.814), while ChatGPT-off-the-shelf lags behind (*i.e.*, 0.711 and 0.690). This confirms that the rules generated by AUTO SIGMA capture the critical context and tactics of each campaign rather than producing generic or poorly aligned detections.

Table E.1: Average results across APT groups for each evaluated model.

APT	Model	IoC Coverage	MITRE Coverage	Relevancy	Validity
APT41	AUTOSigma Llama (Cloud-based)	95.2	90.8	76.5	100
	AUTOSigma OpenAI (Cloud-based)	95.6	91.2	76.4	100
	AUTOSigma Lily (Local-based)	56.0	70.8	73.8	76
	ChatGPT-off-the-shelf	17.6	20.2	71.8	97
APT29	AUTOSigma Llama (Cloud-based)	86.0	91.6	81.6	100
	AUTOSigma OpenAI (Cloud-based)	87.0	91.6	81.4	100
	AUTOSigma Lily (Local-based)	62.4	63.6	76.1	75
	ChatGPT-off-the-shelf	28.0	29.4	69.0	93
APT28	AUTOSigma Llama (Cloud-based)	89.5	92.9	81.4	100
	AUTOSigma OpenAI (Cloud-based)	88.6	91.8	81.6	100
	AUTOSigma Lily (Local-based)	63.8	63.4	75.1	63
	ChatGPT-off-the-shelf	14.8	25.8	71.1	87
Average	AUTOSigma Llama (Cloud-based)	90.2	91.7	79.8	100
	AUTOSigma OpenAI (Cloud-based)	90.4	91.5	79.8	100
	AUTOSigma Lily (Local-based)	60.7	65.9	75.0	71.7
	ChatGPT-off-the-shelf	20.1	25.1	70.6	92.3

Appendix F

Proof of Concept (PoC) (ARMS)

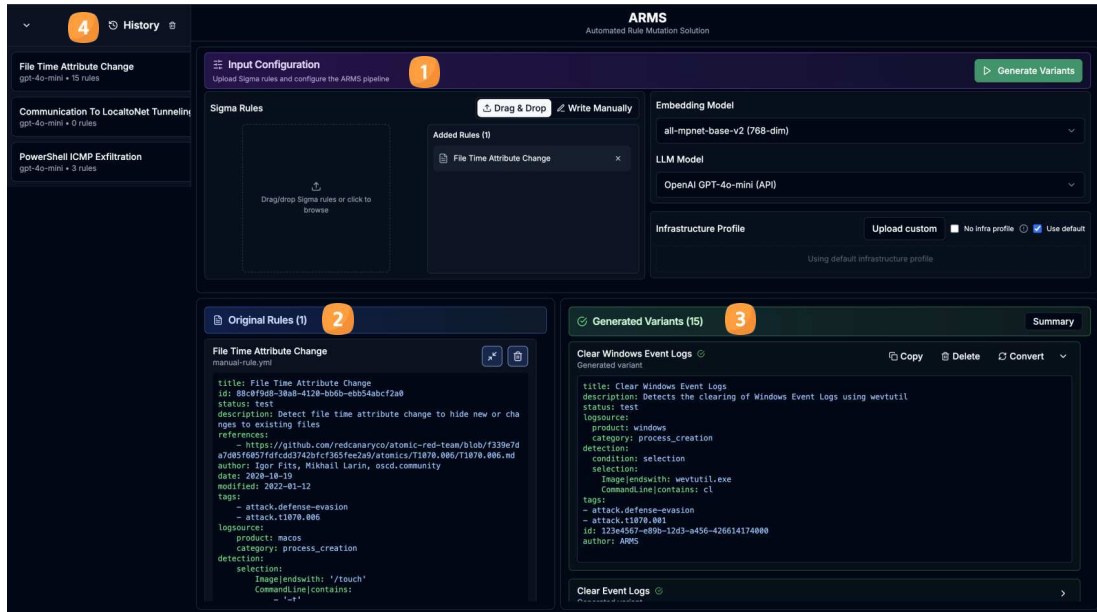


Figure F.1: Screenshot of the ARMS PoC interface.

Fig. F.1 presents the Proof of Concept interface we developed for ARMS, highlighting the end-to-end workflow of the ARMS. The UI is structured into four main sections, visually indicated by yellow rounded-square icons:

- 1 Input Configuration:** Users select the LLM model (local or cloud-based), the embedding model, and upload or paste the sigma rule to initiate variant generation.
- 2 Original Rules:** The bottom left panel visualizes the original rules the user has entered

to be able to compare with the related variants.

3 Generated Variants: This area shows the detailed variant rules which ARMS has generated. Users can convert the rule to a specific SIEM query, copy, and delete a certain rule.

4 History: The final section displays previous generation history for an easy access to the previous outputs.

Bibliography

- [1] W. P. M. III *et al.*, “An Interview Study on Third-Party Cyber Threat Hunting Processes in the U.S. Department of Homeland Security,” in *USENIX Security*, 2024.
- [2] C. Glyer *et al.*, “This Is Not a Test: APT41 Initiates Global Intrusion Campaign Using Multiple Exploits,” <https://www.mandiant.com/resources/blog/apt41-initiates-global-intrusion-campaign-using-multiple-exploits>, 2024.
- [3] Mandiant, “Double Dragon APT41, a dual espionage and cyber crime operation,” <https://www.mandiant.com/sites/default/files/2022-02/rt-apt41-dual-operation.pdf>, 2022.
- [4] CrowdStrike, “Global Threat Report,” <https://www.crowdstrike.com/global-threat-report/>, 2024.
- [5] European Union Agency for Cybersecurity (ENISA), “Enisa threat landscape 2024,” Tech. Rep., Sep. 2024, accessed 2026-03-03. [Online]. Available: https://securitydelta.nl/media/com_hsd/report/690/document/ENISA-Threat-Landscape-2024.pdf
- [6] Canadian Centre for Cyber Security, “National cyber threat assessment 2025–2026,” Tech. Rep., Oct. 2024, accessed 2026-03-03. [Online]. Available: <https://www.cyber.gc.ca/sites/default/files/national-cyber-threat-assessment-2025-2026-e.pdf>
- [7] Microsoft, “Microsoft digital defense report 2024: Executive summary,” Tech. Rep., 2024, accessed 2026-03-03. [Online]. Available: https://cdn-dynmedia-1.microsoft.com/is/content/microsoftcorp/microsoft/final/en-us/microsoft-brand/documents/Exec%20Summary_2024%20Microsoft%20Digital%20Defense%20Report.pdf

- [8] NIST Computer Security Resource Center, “Threat intelligence report,” 2026, accessed 2026-03-03. [Online]. Available: https://csrc.nist.gov/glossary/term/threat_intelligence_report
- [9] C. Johnson, L. Badger, D. Waltermire, J. Snyder, and C. Skorupka, “Nist sp 800-150: Guide to cyber threat information sharing,” NIST, Tech. Rep., 2016, accessed 2026-03-03. [Online]. Available: <https://nvlpubs.nist.gov/nistpubs/specialpublications/nist.sp.800-150.pdf>
- [10] M. Suominen *et al.*, “Cyber threat intelligence management: Tools, techniques, and best practices,” *Cybersecurity Science*, 2024.
- [11] SigmaHQ, “About sigma,” 2025, accessed 2026-03-03. [Online]. Available: <https://sigmahq.io/docs/guide/about.html>
- [12] —, “Main sigma rule repository,” GitHub repository, 2026, accessed 2026-03-03. [Online]. Available: <https://github.com/SigmaHQ/sigma>
- [13] —, “Getting started,” 2025, accessed 2026-03-03. [Online]. Available: <https://sigmahq.io/docs/guide/getting-started.html>
- [14] B. E. Strom, A. Applebaum, D. P. Miller, K. Nickels, A. Pennington, and C. Thomas, “MITRE ATT&CK: Design and philosophy,” The MITRE Corporation, Tech. Rep., 2020, accessed 2026-03-03. [Online]. Available: https://attack.mitre.org/docs/ATTACK_Design_and.Philosophy_March_2020.pdf
- [15] G. Husari *et al.*, “Ttpdrill: Extracting threat techniques and tactics from cyber threat reports,” *Journal of Cyber Threat Intelligence*, 2017.
- [16] L. Gao *et al.*, “Threatraptor: Extracting indicators of compromise and threat behavior from cti reports,” in *IEEE Symposium on Security and Privacy*, 2021, pp. 300–315.
- [17] T. Alam *et al.*, “LADDER: Automated Extraction of Attack Patterns from CTI Reports,” in *Proceedings of the ACM Conference on Security and Privacy*, 2024, pp. 250–265.
- [18] W. Zhang *et al.*, “A systematic review of large language models in cybersecurity,” *ACM Computing Surveys*, 2024.

- [19] M. A. Ferrag *et al.*, “Generative ai and large language models for cybersecurity applications,” *IEEE Transactions on Cybersecurity*, 2024.
- [20] D. Schwartz *et al.*, “LLMCloudHunter: Harnessing LLMs for Automated Extraction of Detection Rules from Cloud-Based CTI,” in *ACM WWW*, 2025.
- [21] M. Xu *et al.*, “Intelex: A llm-driven attack-level threat intelligence extraction framework,” <https://arxiv.org/abs/2412.10872>, 2024.
- [22] OpenAI, “Gpt-4 technical report,” OpenAI, Tech. Rep., 2023. [Online]. Available: <https://cdn.openai.com/papers/gpt-4.pdf>
- [23] R. Uetz, M. Herzog, L. Hackländer, S. Schwarz, and M. Henze, “You cannot escape me: Detecting evasions of SIEM rules in enterprise networks,” in *USENIX Security Symposium (Winter Prepublication)*, 2023. [Online]. Available: <https://www.usenix.org/system/files/sec23winter-prepub-112-uetz.pdf>
- [24] W. P. Maxam III and J. C. Davis, “An interview study on third-party cyber threat hunting processes in the U.S. department of homeland security,” in *33rd USENIX Security Symposium*, 2024. [Online]. Available: <https://www.usenix.org/system/files/usenixsecurity24-maxam.pdf>
- [25] A. Shukla *et al.*, “RuleGenie: Large Language Model Assisted SIEM Rule Optimization,” *arXiv preprint arXiv:2505.06701*, 2025.
- [26] Y. Chen *et al.*, “You Cannot Escape Me: Detecting and Mitigating Evasion Attacks against Detection Rules,” in *Proceedings of the USENIX Security Symposium*, 2023.
- [27] G. González-Granadillo *et al.*, “Security Information and Event Management (SIEM): Analysis, Trends, and Usage in Critical Infrastructures,” 2021. [Online]. Available: <https://www.mdpi.com/1424-8220/21/14/4759>
- [28] M. Roddie *et al.*, *Practical Threat Detection Engineering: A hands-on guide to planning, developing, and validating detection capabilities*. Packt Publishing, 2023.
- [29] S. Alaeifar *et al.*, “Current challenges and future directions in cyber threat intelligence sharing,” *Journal of Cybersecurity Research*, 2024.

- [30] N. Satvat *et al.*, “Extractor: A framework for extracting structured threat intelligence from cti reports,” *Cybersecurity Intelligence Review*, 2021.
- [31] T. Satyapanich *et al.*, “Casie: Extracting cybersecurity event information from text,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.
- [32] I. Sarhan *et al.*, “Open-cykg: An open cyber threat intelligence knowledge graph,” *Knowledge-Based Systems*, 2021.
- [33] J. Park *et al.*, “SecIE: A Full-Stack Information Extraction System for Cybersecurity Intelligence,” in *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 2022.
- [34] J. Liu *et al.*, “Tricti: an actionable cyber threat intelligence discovery system via trigger-enhanced neural network,” *Cybersecurity*, 2022.
- [35] S. Shah *et al.*, “Cyberentrel: Joint extraction of cyber entities and relations using deep learning,” *Computers & Security*, 2024.
- [36] M. D. Purba *et al.*, “Extracting actionable cyber threat intelligence from twitter stream,” in *IEEE International Conference on Cyber Security and Resilience (CSR)*, 2023.
- [37] X. Liu *et al.*, “A study on chatgpt for industry 4.0: Background, potentials, and preliminary applications,” *Journal of Industrial Information Integration*, 2023.
- [38] N. Rastogi *et al.*, “Actionable cyber threat intelligence using knowledge graphs and large language models,” *arXiv preprint arXiv:2407.02528*, 2023. [Online]. Available: <https://arxiv.org/html/2407.02528v1>
- [39] P. A. Gandhi *et al.*, “Shield: Apt detection and intelligent explanation using llm,” *arXiv preprint arXiv:2502.02342*, 2025.
- [40] F. Ahmadou *et al.*, “Automated attack testflow extraction from cyber threat report using bert for contextual analysis,” <https://arxiv.org/abs/2507.07244>, 2025.
- [41] W. Zhang *et al.*, “Few-shot learning of ttps classification using large language models,” 2023.

- [42] Fengrui and Du, “Few-shot learning of ttps classification using large language models,” *arXiv preprint arXiv:2401.0372*, 2024.
- [43] B. E. Strom *et al.*, “MITRE ATT&CK: Design and Philosophy,” *MITRE Technical Report*, 2018.
- [44] L. Li *et al.*, “Automated discovery and mapping att&ck tactics and techniques for unstructured cyber threat intelligence,” *Computers & Security*, 2024.
- [45] A. Basel *et al.*, “Smet: Semantic mapping of cve to att&ck and its application to cybersecurity,” in *Data and Applications Security and Privacy XXXVII*, 2023.
- [46] CVE Program, “Cve: Common vulnerabilities and exposures,” <https://www.cve.org/>, 2025.
- [47] L.-H. Lin *et al.*, “Attack tactic identification by transfer learning of language model,” *arXiv 2209.00263*, 2022.
- [48] S. Community, “Sigma: Generic Signature Format for SIEM Systems,” <https://github.com/SigmaHQ/sigma>, 2024.
- [49] SigmaHQ, “SigmaCLI: The Sigma Command Line Interface,” 2024. [Online]. Available: <https://github.com/SigmaHQ/sigma-cli>
- [50] G. Pham *et al.* (2025) SigmaGen: AI-Powered ATT&CK-Mapped Threat Detection with Sigma Rules. [Online]. Available: <https://blogs.night-wolf.io/sigmagen-ai-powered-attck-mapped-threat-detection-with-sigma-rules>
- [51] P. Lewis *et al.*, “Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks,” in *Advances in Neural Information Processing Systems*, vol. 33, 2020.
- [52] R. Kurniawan *et al.*, “CyKG-RAG: Knowledge Graph Enhanced Retrieval-Augmented Generation for Cybersecurity,” in *CEUR Workshop Proceedings*, 2024.
- [53] Nightwolf Research, “SigmaGen: Automated Generation of Sigma Rules from Threat Intelligence,” 2025, research system.

- [54] H. Wang *et al.*, “Rulepilot: An llm-powered agent for security rule generation,” 2025. [Online]. Available: <https://arxiv.org/abs/2511.12224>
- [55] Falcon Research Group, “FALCON: Automated Signature Generation for Threat Detection,” 2025, research system.
- [56] J. Achiam *et al.*, “GPT-4 Technical Report,” *arXiv preprint arXiv:2303.08774*, 2023.
- [57] J. Gu *et al.*, “A survey on llm-as-a-judge,” <https://arxiv.org/abs/2411.15594>, 2025.
- [58] B. Nour *et al.*, “AUTOMA: Automated Generation of Attack Hypotheses and Their Variants for Threat Hunting Using Knowledge Discovery,” *IEEE Transactions on Network and Service Management*, 2024.
- [59] —, “Threatify: APT Threat Variant Generation Using Graph-Based Machine Learning,” *IEEE Transactions on Network and Service Management*, 2025.
- [60] B. H. S. Durga *et al.*, “Information extraction from text messages using natural language processing,” in *ICCCI*, 2023.
- [61] H. Brådlund *et al.*, “A New HOPE: Domain-agnostic Automatic Evaluation of Text Chunking,” *arXiv 2505.02171*, 2025.
- [62] J. Devlin *et al.*, “Bert: Pre-training of deep bidirectional transformers for language understanding,” 2019. [Online]. Available: <https://arxiv.org/abs/1810.04805>
- [63] T. Zhang *et al.*, “Bertscore: Evaluating text generation with bert,” *arXiv 1904.09675*, 2020.
- [64] K. Papineni *et al.*, “Bleu: a method for automatic evaluation of machine translation,” in *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, 2002.
- [65] T. Zhang *et al.*, “Bertscore: Evaluating text generation with bert,” in *International Conference on Learning Representations (ICLR)*, 2020.
- [66] L. Y. Cheng, “Threats To MITRE (CWD, ATT&CK) AILLM Mapper,” https://github.com/LiuYuancheng/Threats_2_MITRE_AI_Mapper, 2024.

- [67] A. Holtzman *et al.*, “The curious case of neural text degeneration,” <https://arxiv.org/abs/1904.09751>, 2020.
- [68] M. Levi *et al.*, “Cyberpal.ai: Empowering llms with expert-driven cybersecurity instructions,” <https://arxiv.org/abs/2408.09304>, 2024.
- [69] MITRE Corporation, “CAPEC - ATT&CK Mapping,” 2024. [Online]. Available: <https://capec.mitre.org/data/definitions/658.html>
- [70] J. Wei *et al.*, “Chain-of-Thought Prompting Elicits Reasoning in Large Language Models,” in *Advances in Neural Information Processing Systems*, vol. 35, 2022.
- [71] A. V. Aho *et al.*, *Compilers: Principles, Techniques, and Tools*. Addison-Wesley, 2006.
- [72] M. Zhu *et al.*, “AttackGPT: Leveraging Large Language Models for Cyber Threat Intelligence Extraction,” in ACSAC, 2023.
- [73] V. Agarwal, Y. Pei, S. Alamir, and X. Liu, “Codemirage: Hallucinations in code generated by large language models,” *arXiv preprint arXiv:2408.08333*, 2024.
- [74] Red Canary, “Atomic Red Team: A library of simple tests that every security team can execute to test their defenses,” 2024. [Online]. Available: <https://github.com/redcanaryco/atomic-red-team>
- [75] Foundation AI, “Foundation-Sec-8B-Reasoning,” 2024. [Online]. Available: <https://huggingface.co/fdtn-ai/Foundation-Sec-8B-Reasoning>
- [76] Sego Lily Labs, “Lily-Cybersecurity-7B-v0.2,” 2024. [Online]. Available: <https://huggingface.co/segolilylabs/Lily-Cybersecurity-7B-v0.2>
- [77] OpenAI, “GPT-4o Mini,” 2024. [Online]. Available: <https://platform.openai.com/docs/models/gpt-4o-mini>
- [78] Sentence Transformers, “all-mpnet-base-v2,” 2021. [Online]. Available: <https://huggingface.co/sentence-transformers/all-mpnet-base-v2>

- [79] H. Saadany *et al.*, “Bleu, meteor, bertscore: evaluation of metrics performance in assessing critical translation errors in sentiment-oriented text,” *arXiv*, 2021.
- [80] A. Pennino *et al.*, “Game over: Detecting and stopping an apt41 operation,” <https://cloud.google.com/blog/topics/threat-intelligence/game-over-detecting-and-stopping-an-apt41-operation>, 2023.
- [81] G. Cloud, “Apt41 initiates global intrusion campaign using multiple exploits,” <https://cloud.google.com/blog/topics/threat-intelligence/apt41-initiates-global-intrusion-campaign-using-multiple-exploits/>, 2024.
- [82] —, “Apt41 us state governments: Threat intelligence insights,” <https://cloud.google.com/blog/topics/threat-intelligence/apt41-us-state-governments/?hl=en>, 2023.
- [83] H. Hiroaki *et al.*, “Earth Baku An APT Group Targeting Indo-Pacific Countries With New Stealth Loaders and Backdoor,” https://documents.trendmicro.com/assets/white_papers/wp-earth-baku-an-apt-group-targeting-indo-pacific-countries.pdf, 2021.
- [84] ANSSI (CERT-FR), “Campagnes d’attaques du mode opératoire APT28 depuis 2021,” <https://cert.ssi.gouv.fr/uploads/CERTFR-2023-CTI-009.pdf>, 2023.
- [85] ClearSky Cyber Security Research Team, “Doppelgänger NG: Russian Cyberwarfare campaign,” https://www.clearskysec.com/wp-content/uploads/2024/02/DoppelgangerNG_ClearSky.pdf, 2024.
- [86] Insikt Group (Recorded Future), “BlueDelta Exploits Ukrainian Government Roundcube Mail Servers to Support Espionage Activities,” <https://go.recordedfuture.com/hubfs/reports/cta-2023-0620.pdf>, 2023.
- [87] N. Khadgi and U. Thapa, “APT28: Inside Forest Blizzard’s New Arsenal,” <https://www.logpoint.com/wp-content/uploads/2024/06/logpoint-etpr-forest-blizzard.pdf>, 2024.
- [88] Maverits Cybersecurity Center, “APT28, the long hand of Russian interests,” <https://www.maverits.com/post/apt28-the-long-hand-of-russian-interests>, 2023.

- [89] CrowdStrike Intelligence Team, “Observations from the stellarparticle campaign,” <https://www.crowdstrike.com/blog/observations-from-the-stellarparticle-campaign/>, 2023.
- [90] National Coordination Center for Cybersecurity at the NSDC of Ukraine, “APT29 attacks Embassies using CVE-2023-38831,” https://rnbo.gov.ua/files/2023_YEAR/CYBERCENTRE/november/APT29attacksEmbassiesusingCVE-2023-38831-reporten.pdf, 2023.
- [91] Recorded Future Insikt Group, “Bluebravo adapts to target diplomatic entities with graphical-proton malware,” <https://www.recordedfuture.com/research/bluebravo-adapts-to-target-diplomatic-entities-with-graphicalproton-malware>, 2024.
- [92] Mandiant- Google Cloud, “Not so cozy: An uncomfortable examination of a suspected apt29 phishing campaign,” <https://www.fireeye.com/blog/threat-research/2018/11/not-so-cozy-an-uncomfortable-examination-of-a-suspected-apt29-phishing-campaign.html>, 2018.
- [93] “Russian GRU Conducting Global Brute Force Campaign to Compromise Enterprise and Cloud Environments,” https://media.defense.gov/2021/Jul/01/2002753896/-1/-1/0/CSA_GRU_GLOBAL_BRUTE_FORCE_CAMPAIGN_UOO158036-21.PDF, 2021.