From Texts to Rules: Generating Sigma Rules with Large Language Models from Cyber Threat Reports

Yongxin Cai[1], Jing Qiu[1,2,*], Qingming Li[3], Du Cheng[4], Lei Chen[5]
[1]*Guangzhou University* [2]*Pengcheng Laboratory* [3]*Zhejiang University* [4]*Tsinghua University*
[5]*Hong Kong University of Science and Technology (Guangzhou)*
[*]*Corresponding author*

Abstract

Cyber Threat Reports (CTRs) deliver actionable intelligence essential for security systems detection rules. Large language models (LLMs) could serve as a bridge for CTRs-to-Rules translation through parsing and generation capabilities. However, the semantic disconnect and domain-specific constraints between high-level abstractions in CTRs and low-level machine semantics in rules fundamentally impede accurate detection rules generation.

In this paper, we demonstrate that shell commands in CTRs can be effectively converted into Sigma detection rules for security systems. To this end, we propose **SIGMERGE**, an end-to-end framework that generates Sigma rules from texts of CTRs by constructing a semantic intermediate layer as a bridge. The **SIGMERGE** framework hierarchically organizes three modules by descending semantic levels: (1) The Information extraction module, high-level, utilizes a multi-subsequence algorithm and a fine-tuned domain-specific LLM, enabling accurate MITRE ATT&CK tactics, techniques, and procedures (TTPs) and command extractions; (2) The Attack description generation module, intermediate-level, employs preference optimization tuning with closed-loop self-validation to mitigate the semantic disconnect; (3) The Sigma rule generation module, machine-level, leverages a parameter-optimized retrieval algorithm to address domain-specific constraints. We constructed 7 datasets for training and conducted extensive experiments. To validate **SIGMERGE**, we evaluated it using 23 metrics against 16 baselines and 13 LLMs, and conducted 10 case studies integrated with real security systems to demonstrate both effectiveness and efficiency. Moreover, **SIGMERGE** has already contributed 4 novel Sigma rules to the official repository, all of which have been formally accepted.

1 Introduction

Cyber Threat Reports (CTRs) are central to modern cyber defense, documenting attack intents and behaviors that support timely identification of emerging threats. Prior research has exploited CTR text for tasks such as entity recognition [11,13,73], attack graph construction [29,41,61], and MITRE ATT&CK tactics, techniques, and procedures (TTPs) mapping [14,26,33]. However, these approaches share a critical limitation: their outputs primarily serve human analysts, such as through association analysis and visualization, rather than security systems. This human-centric orientation leaves a crucial gap: existing methods fail to produce machine-level detection rules.

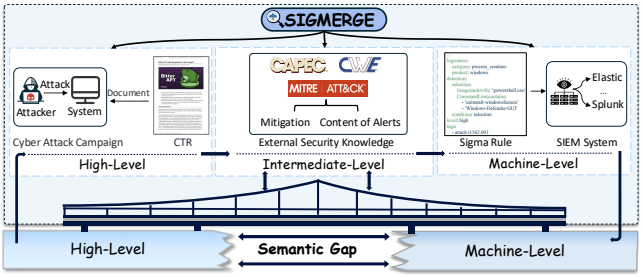

Figure 1: **SIGMERGE** is guided by the principle of semantic descent and operates across three levels: high-level unstructured CTR text, intermediate-level external security knowledge, and machine-level Sigma rules. The generated rules are directly executable in SIEM systems for real-time detection.

Recent advances in large language models (LLMs) have opened new opportunities for converting machine-level detection rules directly from CTRs [55, 65]. Empirical studies [11,72] show that mainstream LLMs can interpret attack behaviors described in CTRs with simple prompts and exhibit preliminary competence in handling rules such as Yara, Snort, and Sigma. However, direct CTRs-to-Rules translation with LLMs reveals significant constraints: (1) the unsuitability of certain rule types for generation from CTRs; (2) the semantic disconnect between CTRs and rules; (3) detection rules have domain-specific structural and semantic requirements.

Rule Suitability. The three main types of detection rules in current practice are Yara, Snort, and Sigma. Among them, Yara and Snort are not suitable for generation from CTR text.

Our analysis of 1,239 open-source Yara rules [69] revealed that only two contained all necessary detection details in the corresponding reports, demonstrating that CTR text alone is insufficient to reconstruct Yara. Snort, on the other hand, primarily relies on network indicators (e.g., IP addresses, protocols), which are short-lived and easily altered by adversaries. In contrast, Sigma rules are designed to detect malicious activity in security logs, particularly shell command executions. Notably, more than 94% of CTR in our dataset contain complete shell commands with contextual descriptions, which confirms Sigma's strong suitability as the target rule format.

Semantic Disconnect. Despite Sigma's suitability, high-quality rule generation remains challenging. The absence of learnable mappings between high-level CTR text (i.e., unstructured, human-readable narratives) and machine-level detection rules (i.e., Sigma rules directly usable by SIEM) creates a semantic gap. This gap arises because identical attack objectives can often be expressed through diverse command variants, where differences in operating systems, software versions, and execution environments yield functionally equivalent implementations. Addressing this gap is essential for improving the robustness of Sigma rules.

Domain-Specific Constraints. While handling the semantic gap can improve robustness, generating usable Sigma rules requires adherence to the domain-specific structural and semantic constraints inherent to Sigma rules. Specifically, these constraints concern how malicious command parameters should be represented and combined within the rule, as well as how detection operators should be selected. Any mistake in these aspects directly affects the rule's usability.

In this paper, we propose **SIGMERGE**, the first end-to-end framework that converts CTR text into Sigma rules directly deployable on Security Information and Event Management (SIEM) systems. As illustrated in Figure 1, **SIGMERGE** introduces an intermediate-level (external security knowledge) representation that links high-level CTR text with machine-level Sigma rules. The framework achieves this by addressing the following challenges.

- **Challenge I: How to accurately extract behavior intents and shell command texts from CTRs to serve as key inputs for generating high-quality Sigma rules?**

To capture attack intent in CTRs, we first extract TTPs, which serve as essential inputs for downstream tasks such as linking to external security knowledge. However, existing TTPs extraction methods often fail to capture the compositional semantics of attack elements. For example, as shown in Figure 3, the input sentence contains multiple components—Actor (e.g., Threat actor), Tool (e.g., PowerShell), and Action (e.g., Uninstall). Accurate TTPs extraction requires integrating their latent semantic representations. To address this issue, we introduce a multi-subsequence algorithm that models compositional semantics among attack elements, thereby improving TTPs extraction accuracy. In addition, shell commands in CTRs are syntactically complex and often obfuscated by special characters or redundant spaces, which complicates boundary identification. To overcome this challenge, we design a shell command extraction module that builds a fine-tuning dataset and leverages few-shot prompting to achieve precise shell command extraction.

- **Challenge II: How to acquire and associate external knowledge to effectively bridge the semantic gap between the CTR command text and Sigma rules?**

Shell command semantics are highly diverse, and deriving Sigma rules solely from literal command strings often results in brittle detection logic. For example, a rule generated only from the command *powershell.exe Uninstall-WindowsFeature* may miss variants such as *powershell.exe Remove-WindowsFeature*, which achieve the same effect with different syntax. To overcome this limitation, we shift the focus from literal command strings to the attacker's intent, capturing the behavior rather than the syntax. Building on this idea, we design an attack description generation module that leverages external security knowledge bases. It incorporates extracted TTPs into four knowledge types: *CWE* (i.e., Common Weakness Enumeration), *CAPEC* (i.e., Common Attack Pattern Enumeration and Classification), *CoA* (i.e., Content of Alerts), and *Mitigation* via LLMs, generating structured attack descriptions. [1] These descriptions are enriched by fusing command context with customized templates, fine-tuned using Direct Preference Optimization (DPO), and validated with a self-correction mechanism. The resulting textual representations bridge the semantic gap, providing interpretable foundations for generating high-quality Sigma rules.

- **Challenge III: How to effectively integrate domain expertise into a general-purpose LLM to enable accurate and efficient generation of high-quality Sigma rules?**

General-purpose LLMs often fail to generate accurate Sigma rules due to limited domain knowledge, particularly the structure of Sigma rules and field definitions. To address this, we design a two-stage Sigma rule generation module: a knowledge-enhanced integration phase and a rule generation phase. In the first phase, the LLM augments golden Sigma rules with contextual security knowledge and combines them with textual attack descriptions to build parameterized documents stored in a retrieval database. In the second phase, the system retrieves rule parameters by matching input commands against the database. It then injects the retrieved parameters into the LLM's adapter layers to produce syntax-compliant Sigma rules. This design equips lightweight LLMs

[1] Here, the CoA field corresponds to the *Detection Strategy* attribute in MITRE ATT&CK, which specifies observable indicators. The Mitigation field corresponds to the *Mitigations* attribute, which provides defensive strategies. See https://attack.mitre.org/techniques/T1554/ for a real-world example.

with domain expertise. Our experiments show that, compared to DeepSeek-V3 (a commercial LLM), our method improves rule completeness by 33.17% and achieves a 2.85× speedup.

To address the three challenges, we design **SIGMERGE** with three modules that progressively handle **information extraction** (high-level), **attack description generation** (intermediate-level), and **Sigma rule generation** (machine-level). Our prototype demonstrates consistent gains across all modules. In TTPs extraction, it yields a relative *F1* improvement of 75.50% for tactics and 86.52% for techniques over baselines. For command extraction, it achieves an *F1* score of 98.93%. In attack description generation, it reaches 89.76% semantic similarity with ground-truth references. For Sigma rule generation, it improves rule completeness by 33.17% over DeepSeek-V3. These results highlight the effectiveness and practicality of **SIGMERGE**.

We summarize our contributions as follows:

- We present **SIGMERGE**, the first framework to systematically analyze CTR-to-Sigma suitability and innovatively introduce the semantic descent approach through three semantic levels for high-quality Sigma rule generation.
- We design metrics for Sigma rules that jointly assess structural correctness and field-level accuracy, offering the first benchmark for this task.
- **SIGMERGE** builds 7 annotated datasets and undergoes large-scale evaluation across 23 metrics, 16 baselines, and 13 LLMs.
- We validate **SIGMERGE** on ten reports by reproducing their attack scenarios, logging system events, and evaluating the generated Sigma rules for both detection accuracy and runtime efficiency.
- **SIGMERGE** has contributed 4 novel Sigma rules to the official repository, all of which have been accepted, demonstrating practical utility for real-world defense.

2 Background and Motivation

2.1 CTRs-to-Sigma Translator

Background. CTRs document real intrusion activity with analyst-verified behavioral detail [17, 30, 42], capturing attacker actions, executed commands, and operational intent across multiple stages of an intrusion campaign. GreyNoise [19] reports that 80% of malicious activity surges precede related CVE disclosures by up to six weeks, showing that CTR often record early indicators of emerging threats. However, because CTRs are written in unstructured natural language, their behavioral details remain difficult to convert into actionable rules. Moreover, each rule type relies on different factors: Yara relies on code-level signatures rarely present in CTRs, and Snort is dominated by short-lived indicators of compromise (IoCs) such as IP addresses. In contrast, Sigma is explicitly designed for behavioral detection in system logs, aligning naturally with CTR content. This makes CTR-to-Sigma translation not only feasible in practice, but also crucial for strengthening proactive defense.

Motivation. Through our study of CTRs, we identified three key phenomena that motivate **SIGMERGE**, which addresses them through a semantic descent perspective for bridging CTRs and Sigma rules.

(1) *High-Level.* CTRs frequently embed complete shell commands in descriptive text, making command extraction a natural first step toward rule construction. Yet shell commands appear in diverse forms across operating systems, software versions, and even with obfuscation, which complicates accurate extraction. Moreover, a command's maliciousness often depends on its contextual TTPs, since the same command may indicate benign administration or malicious intent. Thus, precise command extraction is the basis of constructing rules, while TTPs extraction is the basis of constructing high-quality rules, together addressing Challenge I.

(2) *Intermediate-Level.* We observed that the same attack objective can often be realized through diverse command variants arising from operating system differences, version changes, or obfuscation. For example, removing Windows Defender can be achieved through `reg.exe`, the PowerShell cmdlet `Remove-Item`, or its alias `ri`, all achieving the same purpose despite substantial syntactic differences. Relying solely on command text therefore risks missing equivalent behaviors and produces brittle rules. To address this, we turn to external security knowledge bases, which provide complementary perspectives on vulnerabilities, attack patterns, observables, and defenses. These dimensions allow us to move beyond surface syntax and supply the context necessary for robust rule generation, thereby addressing Challenge II.

(3) *Machine-Level.* We observed that LLMs without domain knowledge grounding frequently produce Sigma rules that are either incomplete or syntactically invalid, particularly when training data are scarce or when commands appear in multiple variants. For example, LLMs may combine invalid operators (e.g., `contains|all`), producing non-deployable Sigma rules. To overcome this, we design a retrieval-augmented generator (RAG) that integrates domain expertise, ensuring rules are both accurate and efficient. This directly addresses Challenge III.

2.2 Shell Commands Analysis

Shell Commands. A shell command is a text-based instruction executed by command-line interpreters, consisting of a utility, options, and parameters. For example, *bash -c "rm -rf /tmp/logs/*"* may indicate routine log cleanup in one context but forensic evidence deletion in another. As primary artifacts of the execution tactic in the MITRE ATT&CK framework, shell commands frequently appear in CTRs. However, their structural complexity, syntactic variability, and context-dependent semantics make it difficult to precisely infer intent.

This challenge necessitates intermediate representations that bridge high-level descriptions and machine-level executable rules.

Tactics, Techniques, and Procedures (TTPs). Analyzing malicious shell commands requires understanding both their semantics and their TTPs contexts. The adopted taxonomy is MITRE ATT&CK, a knowledge base built on real-world intrusions. Since the intent of a command is usually determined by its preceding TTP, analyzing commands in isolation yields ambiguous results. Threat reports often describe the attacker's prior steps before presenting a specific command, which defines its operational purpose. For example, *certutil -decode* may denote benign administration or payload retrieval, depending on whether it follows phishing (T1566) or exploitation of public-facing applications (T1190). Therefore, identifying TTPs in CTRs is crucial for interpreting shell commands. To this end, **SIGMERGE** introduces a hierarchical TTP extractor that correlates command semantics with ATT&CK.

Attack Descriptions. While TTPs capture attacker behaviors, they lack defensive dimensions necessary for automated Sigma rule translation. External knowledge bases complement this gap: *CWE* explains vulnerability causes (why), *CAPEC* defines attack patterns (what), *CoA* identifies observables (where), and *Mitigation* frameworks prescribe defenses (how).

2.3 A Real-World Motivating Example

This section demonstrates **SIGMERGE**'s ability to process authentic CTRs and translate malicious commands into executable Sigma rules. In the analyzed case, the attacker issued a command to disable Windows Defender in order to evade default protections. Figure 2 illustrates how **SIGMERGE** transforms this high-level CTR description through three coordinated modules into a machine-level Sigma rule, which can then be compiled into search queries for SIEM platforms such as Splunk and Elastic.

3 Threat Model

Following our observations of real-world CTR workflows, we formalize the setting of **SIGMERGE** through the goals and capabilities of three entities: the attacker, the security analyst, and the defender.

Attacker. We assume an attacker who seeks to compromise a target environment and maintain control throughout the intrusion lifecycle. The attacker can conduct reconnaissance, obtain initial access through common techniques (e.g., valid accounts, phishing, exploiting public-facing services, etc.), and execute commands that alter host or network state, leaving traces that can be captured by host or network monitoring mechanisms. However, the attacker cannot tamper with CTRs or interfere with the internal workflows of analysts or defenders.

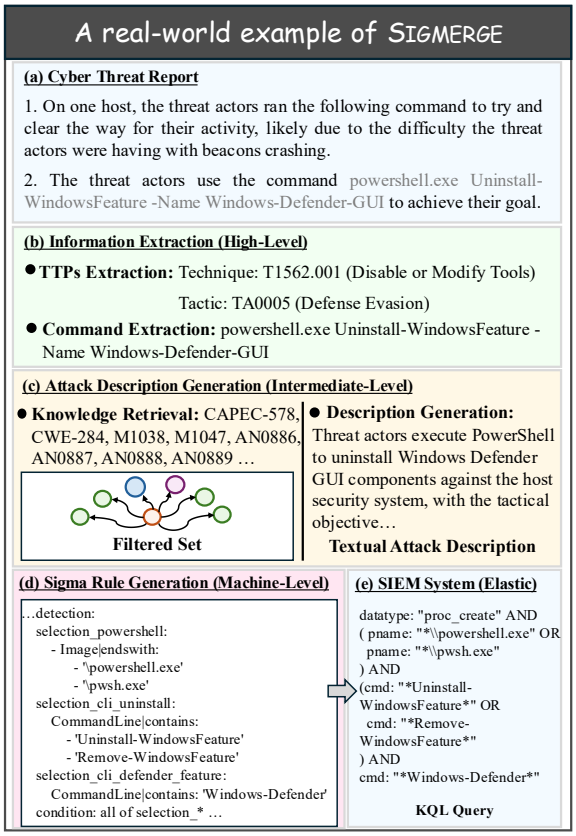

Figure 2: A real-world example of **SIGMERGE** illustrates its three-layer design and the flow of inputs and outputs across these layers. The resulting Sigma rule is then translated into executable queries for the Elastic SIEM platform.

Security Analyst. The analyst investigates the incident using standard forensic procedures and reconstructs the adversary's behavior from diverse evidence sources, including command artifacts, malware analysis, host-level traces, and network activity. Following prior work on automated CTR processing, we assume the CTR produced by analysts is trustworthy and faithfully reflects the adversary's actions.

Defender. The defender receives the finalized CTR and aims to deploy actionable detection logic in a SIEM system. The defender does not reconstruct the attack or manually craft Sigma rules. Instead, the defender provides the CTR to **SIGMERGE**, which processes the command evidence in the report and generates Sigma rules that can be translated into platform-specific queries such as Splunk or Elastic detections.

4 SIGMERGE: Detailed Construction

This section presents **SIGMERGE**, a framework for generating Sigma rules from CTR. As shown in Figure 3, the pipeline comprises three phases: *information extraction*, *attack description generation*, and *Sigma rule generation*.

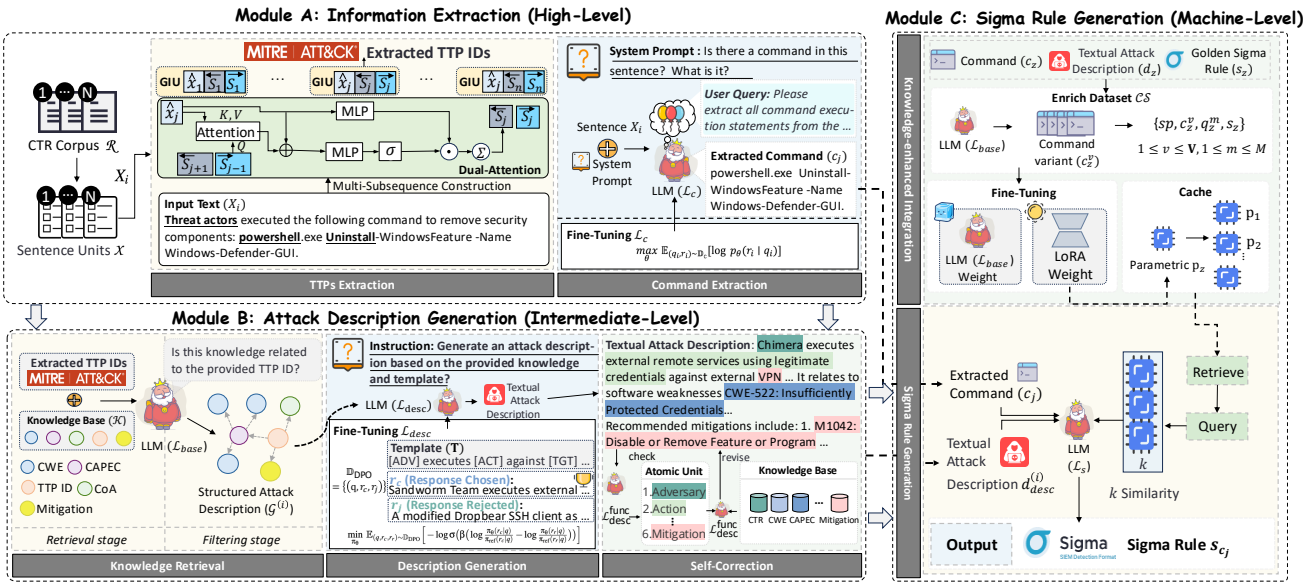

Figure 3: **SIGMERGE** architecture, with an illustrative input example. The prompt terms only simply indicate the task's intent. For the specific prompt terms, refer to the **SIGMERGE** artifacts for details.

4.1 Problem Formulation and Overview

Information Extraction Problem Formulation. A cyber threat report $\mathcal{R}$ consists of sentences $\mathcal{X} = \{X_1, \ldots, X_N\}$, where N is the total number of sentences. For each sentence X_i, we identify its TTPs, denoted as $Y_{\text{ta}}^{(i)} \subseteq \mathcal{Y}_{\text{ta}}$ and $Y_{\text{te}}^{(i)} \subseteq \mathcal{Y}_{\text{te}}$, where $\mathcal{Y}_{\text{ta}}$ and $\mathcal{Y}_{\text{te}}$ are the predefined tactic and technique sets from MITRE ATT&CK. We also extract all shell commands in X_i using a fine-tuned LLM $\mathcal{L}_c$, yielding a command set $\mathcal{C}^{(i)}$, where each $c_j \in \mathcal{C}^{(i)}$ is a contiguous substring of X_i.

Attack Description Generation Problem Formulation. Given a sentence X_i and its prior extracted techniques $Y_{\text{te}}^{(\le i)}$, the goal is to generate a textual attack description $d_{\text{desc}}^{(i)}$. First, a pre-trained LLM $\mathcal{L}_{\text{base}}$ retrieves entities from external security knowledge bases $\mathcal{K}$, which contain *CoA*, *CWE*, *CAPEC*, and *Mitigation* information indexed by technique IDs. Together with $Y_{\text{te}}^{(\le i)}$, the selected knowledge forms the structured attack description $\mathcal{G}^{(i)}$. Second, a fine-tuned LLM $\mathcal{L}_{\text{desc}}$ generates a description $d_{\text{gen}}^{(i)}$ on $\mathcal{G}^{(i)}$, X_i, and a template $\mathbf{T}$ (Figure 4). Finally, a self-correction model $\mathcal{L}_{\text{desc}}^{\text{func}}$, adapted from $\mathcal{L}_{\text{desc}}$, refines $d_{\text{gen}}^{(i)}$ into the final verified description $d_{\text{desc}}^{(i)}$.

Sigma Rule Generation Problem Formulation. Given a command $c_j \in \mathcal{C}^{(i)}$ and its attack description $d_{\text{desc}}^{(i)}$, the goal is to generate the corresponding Sigma rule s_{c_j}. We enhance the base model $\mathcal{L}_{\text{base}}$ with a parametric cache $\mathcal{M}$, built from a command-to-Sigma corpus $\mathcal{CS}$. At generation time, for each input command c_j, we compute similarity scores $\text{sim}(c_j, c_z)$ with all $c_z \in \mathcal{CS}$ and retrieve the top-k nearest entries. These are used to augment $\mathcal{L}_{\text{base}}$, yielding an enhanced model $\mathcal{L}_s$, which generates the Sigma rule s_{c_j}.

4.2 Information Extraction

To address Challenge I, this subsection introduces two modules: TTPs extraction and command extraction.

TTPs Extraction. To capture compositional attack semantics, we design a multi-subsequence modeling framework with bidirectional memory propagation. Firstly, for the i-th input sentence $X_i = (x_1, \ldots, x_n)$, where x_j denotes the j-th token, we first obtain contextual embeddings with BERT:

$$X' = [x'_1, \ldots, x'_n] = \text{BERT}(X_i) \in \mathbb{R}^{n \times d}, \tag{1}$$

where d is the embedding dimension. For each token representation x'_j, we then extract a local subsequence window to capture the semantics of neighboring attack elements:

$$\hat{\mathbf{x}}_j = [x'_{j-k}, \ldots, x'_{j+k}] \in \mathbb{R}^{(2k+1) \times d}, \quad j \in 1, \ldots, n, \tag{2}$$

where $\hat{\mathbf{x}}_j$ subsequence of the j-th token and k is a predefined window size controlling the local context range.

Secondly, we employ a dual-attention mechanism to enhance each token's representation. We propagate compositional semantics by leveraging two memory states from the previous time step: a forward memory $\overrightarrow{\mathbf{m}}_{j-1}$ and a backward memory $\overleftarrow{\mathbf{m}}_{j+1}$. These states serve as queries, while the local subsequence $\hat{\mathbf{x}}_j$ provides the corresponding keys and values. This attention operation updates the memory to $\overrightarrow{\mathbf{m}}_j$ and $\overleftarrow{\mathbf{m}}_j$ for the j-th token x'_j. Then, to retain valuable information from the previous time step, we compute a local summary vector $\mathbf{g}_j = \bigoplus_{\mathbf{x} \in \hat{\mathbf{x}}_j} \mathbf{x}$ from the subsequence $\hat{\mathbf{x}}_j$ and derive a retention signal $\overrightarrow{\mathbf{r}}_j = \sigma(\text{MLP}([\overrightarrow{\mathbf{m}}_j, \mathbf{g}_j]))$, where σ is the sigmoid function. The output state combines the transformed sum-

mary with the gated memory: $\overrightarrow{\mathbf{s}}_j = \mathrm{MLP}(\mathbf{g}_j) + \overrightarrow{\mathbf{r}}_j \odot \overrightarrow{\mathbf{m}}_j$.[2] To stabilize activation magnitudes and preserve weak but informative signals, we apply a gated interaction unit (GIU) with layer normalization and LeakyReLU. Here, the tanh nonlinearity bounds the gated memory values to a normalized range:

$$\begin{aligned}\overrightarrow{\mathbf{s}}'_j &= \mathrm{LeakyReLU}\left(\mathrm{LayerNorm}\left(\sigma(\mathrm{MLP}(\mathbf{g}_j)) \odot \tanh(\overrightarrow{\mathbf{s}}_j)\right)\right), \\ \overleftarrow{\mathbf{s}}'_j &= \mathrm{LeakyReLU}\left(\mathrm{LayerNorm}\left(\sigma(\mathrm{MLP}(\mathbf{g}_j)) \odot \tanh(\overleftarrow{\mathbf{s}}_j)\right)\right).\end{aligned} \tag{3}$$

Finally, the prediction is generated by fusing the original token representation with the contextualized features from both directions: $\mathbf{u}_j = \mathbf{x}_j \oplus \overrightarrow{\mathbf{s}}'_j \oplus \overleftarrow{\mathbf{s}}'_j$. The sentence-level representation is taken as the fused representation of the first token (the `[CLS]` token), denoted as $\mathbf{u}_0$. Separate MLPs then predict tactic and technique distributions directly from this representation: $\mathbf{p}^{(i)}_{\mathrm{ta}} = \mathrm{MLP}_{\mathrm{ta}}(\mathbf{u}_0)$ and $\mathbf{p}^{(i)}_{\mathrm{te}} = \mathrm{MLP}_{\mathrm{te}}(\mathbf{u}_0)$. The final TTP sets are obtained by thresholding the predicted distributions at τ_{ta} and τ_{te}, respectively:

$$\begin{aligned}Y^{(i)}_{\mathrm{ta}} &= \left\{ y_{ta} \in \mathcal{Y}_{\mathrm{ta}} \mid \mathbf{p}^{(i)}_{\mathrm{ta},y_{ta}} > \tau_{\mathrm{ta}} \right\}, \\ Y^{(i)}_{\mathrm{te}} &= \left\{ y_{te} \in \mathcal{Y}_{\mathrm{te}} \mid \mathbf{p}^{(i)}_{\mathrm{te},y_{te}} > \tau_{\mathrm{te}} \right\}.\end{aligned} \tag{4}$$

Here, $Y^{(i)}_{\mathrm{ta}}$ and $Y^{(i)}_{\mathrm{te}}$ denote the sets of tactics and techniques assigned to sentence X_i, respectively.

Command Extraction. General-purpose LLMs struggle with command extraction because real-world commands exhibit irregular syntax and formatting. To address this, we fine-tune a base LLM $\mathcal{L}_{\mathrm{base}}$ into a specialized extractor $\mathcal{L}_{\mathrm{c}}$ (defined in §4.1) using LoRA [22], which yields:

$$\mathcal{C}^{(i)} = \{c_1, \ldots, c_{n_c}\} = \mathcal{L}_{\mathrm{c}}(X_i), \tag{5}$$

where X_i is the input sentence, $\mathcal{C}^{(i)}$ denotes the extracted command set and n_c denotes the number of commands in the set. Training is based on structured triplets ⟨ system prompt, user prompt, target command ⟩, where each example consists of a query $q_i = (sp_i, up_i)$, with sp_i as the system prompt, up_i as the user prompt, and the target command r_i. Formally, we maximize:

$$\max_{\theta} \mathbb{E}_{(q_i,r_i)\sim\mathbb{D}_{\mathrm{c}}} \left[\log p_\theta(r_i \mid q_i)\right], \tag{6}$$

where θ are LoRA parameters and $\mathbb{D}_{\mathrm{c}}$ is the fine-tuning dataset.

To ensure robustness, we construct the dataset $\mathbb{D}_{\mathrm{c}}$, which covers both diverse command types and varied linguistic contexts. Commands are collected from *atomic-red-team*, *metta*, and *PowerShell* repositories, all paired with natural-language descriptions. For sentence variation, we use LLM rewriting to diversify lexical and syntactic expression, vary command positions (beginning, middle, end), and merge multiple commands into single inputs. The full prompts are shown in Appendix A.1.

[2]The backward pass follows a symmetric computation, producing $\overleftarrow{\mathbf{s}}_j$ from $\overleftarrow{\mathbf{m}}_j$ and the same $\mathbf{g}_j$.

4.3 Attack Description Generation

To address Challenge II, this subsection performs attack description generation in three steps.

Knowledge Retrieval. Given the prior extracted techniques $Y^{(\le i)}_{\mathrm{te}}$, we construct a structured attack description $\mathcal{G}^{(i)}$ in two stages: *retrieval* and *filtering*.

Retrieval. We collect all knowledge entries linked to each $y_{te} \in Y^{(\le i)}_{\mathrm{te}}$ from the external knowledge base $\mathcal{K}$ (example for T1562.001 in Appendix A.2), forming the initial set:

$$k^{(i)}_{init} = \bigcup_{y_{te} \in Y^{(\le i)}_{\mathrm{te}}} \mathcal{K}[y_{te}], \tag{7}$$

where $\mathcal{K}[y_{te}]$ includes *CoA*, *CWE*, *CAPEC*, and *Mitigation* associated with y_{te}. This step is context-agnostic, aggregating all related entries without considering the semantics of X_i.

Filtering. We apply $\mathcal{L}_{\mathrm{base}}$ with prompt engineering to refine $k^{(i)}_{init}$ by retaining only entries semantically aligned with X_i (the full prompt is provided in Appendix A.2). The resulting entries form the node set of $\mathcal{G}^{(i)}$, consisting of techniques and their associated knowledge elements.

Description Generation. We generate standardized attack descriptions $d^{(i)}_{\mathrm{gen}}$ from $\mathcal{G}^{(i)}$ and X_i through a textual template $\mathbf{T}$ (detailed in Figure 4), which enforces consistent formatting for downstream reasoning. The *background* field is derived from X_i and its surrounding paragraph to resolve contextual references, while all other fields (e.g., *CoA*, *CWE*, *CAPEC*, and *Mitigation*) are populated from $\mathcal{G}^{(i)}$. The prompt design for generating $d^{(i)}_{\mathrm{gen}}$ is shown in Appendix A.2.

A key challenge is ensuring that $\mathcal{L}_{\mathrm{desc}}$ generates $d^{(i)}_{\mathrm{gen}}$ faithfully following $\mathbf{T}$. To this end, we build a preference dataset $\mathbb{D}_{\mathrm{DPO}} = \{(q, r_c, r_r)\}$, where each query $q = (I, \mathcal{G}^{(i)})$ combines a natural-language instruction I with the input graph $\mathcal{G}^{(i)}$. The preferred response r_c adheres to $\mathbf{T}$, while the rejected response r_r exhibits formatting errors or missing fields.

We fine-tune $\mathcal{L}_{\mathrm{base}}$ on $\mathbb{D}_{\mathrm{DPO}}$ using DPO, directly aligning model outputs with preferences without explicit reward modeling. The optimization objective is:

$$\min_{\pi_\theta} \mathbb{E}_{(q,r_c,r_r)\sim\mathbb{D}_{\mathrm{DPO}}} \left[-\log \sigma\left(\beta\left(\log \frac{\pi_\theta(r_c|q)}{\pi_{\mathrm{ref}}(r_c|q)} - \log \frac{\pi_\theta(r_r|q)}{\pi_{\mathrm{ref}}(r_r|q)}\right)\right)\right], \tag{8}$$

where π_θ is initialized from the base policy π_{ref}, β controls the deviation, and σ is the logistic function. The resulting model is denoted as $\mathcal{L}_{\mathrm{desc}}$, and its operation can be summarized as: $d^{(i)}_{\mathrm{gen}} = \mathcal{L}_{\mathrm{desc}}(X_i, \mathcal{G}^{(i)})$.

Self-Correction. Even with template $\mathbf{T}$, generated descriptions $d^{(i)}_{\mathrm{gen}}$ may hallucinate facts absent from $\mathcal{G}^{(i)}$. We categorize such errors as *extrinsic* (content not in $\mathcal{G}^{(i)}$, e.g., a non-existent CWE) and *intrinsic* (incorrect details, e.g., mislabeling "CWE-1025: Comparison Using Wrong Factors" as "CWE-1025: Buffer Overflow").

Attack Description Template

Background
`[ADV] executes [ACT] against [TGT], with the tactical objective of achieving [OBJ], leveraging [TOOL].`
Content of Alert (CoA)
`This activity can be detected by monitoring [Data Sources] and analyzing the following data components: [Data Components].`
`Data Sources: {Data Source Name}, ...`
`Data Components: N) {Component Name}: {Component Description}`
Weaknesses (CWE)
`Weaknesses: N) [{CWE ID}: {CWE Name} -- {CWE Description}]`
Attack Pattern (CAPEC)
`Attack patterns: N) [{CAPEC ID}: {CAPEC Name}]`
`Descriptions: N) {CAPEC Description}`
Mitigation
`Mitigations: N) [{Mitigation ID}: {Mitigation Name} -- {Mitigation Description}]`

Definitions:
- `ADV`: Attacker/group (e.g., APT41, Lazarus)
- `ACT`: Core attack action (e.g., abuse token privileges)
- `TGT`: Attack target (e.g., account, service, system)
- `OBJ`: Tactical intent (e.g., privilege escalation)
- `TOOL`: Tool/payload description (e.g., BADPOTATO)

Figure 4: Attack description template with field definitions.

To mitigate these errors, we introduce a self-correction module that refines $d_{\text{gen}}^{(i)}$ into a verified description $d_{\text{desc}}^{(i)}$. The module employs $\mathcal{L}_{\text{desc}}^{\text{func}}$, a tool-augmented variant of $\mathcal{L}_{\text{base}}$ fine-tuned via LoRA on the dataset $\mathbb{D}_{\text{SC}}$. Its functionality is summarized as: $d_{\text{desc}}^{(i)} = \mathcal{L}_{\text{desc}}^{\text{func}}\left(d_{\text{gen}}^{(i)}\right)$. Specifically, $\mathcal{L}_{\text{desc}}^{\text{func}}$ parses $d_{\text{gen}}^{(i)}$ into template-defined fields (*Background*, *CoA*, *CWE*, *CAPEC*, *Mitigation*), and invokes dedicated validators for each field. For example, the *CWE* field is checked against $\mathcal{K}_{CWE}$ to verify existence and description consistency. The validators return validated entries, which $\mathcal{L}_{\text{desc}}^{\text{func}}$ leverages to identify inconsistencies and revise the original text. The corrected fields are then reassembled into the final verified description $d_{\text{desc}}^{(i)}$.

4.4 Sigma Rule Generation

Prior studies have demonstrated the effectiveness of RAG for injecting external knowledge into LLMs [28, 58, 59]. To address Challenge III, we base our approach on a ParamCache-based RAG mechanism that integrates Sigma rules and command information directly into model parameters, thereby enhancing knowledge retention and improving downstream rule generation. This subsection consists of two phases: Knowledge-enhanced integration and Sigma rule generation.

Knowledge-enhanced integration. To address the lack of mapping expertise between shell commands and Sigma rules in general LLMs, our approach consists of three stages. First, we construct a retrieval corpus $\mathcal{CS} = (c_z, d_z, s_z)_{z=1}^{Z}$ from the official Sigma GitHub repository, where Z is the number of command-inspection rules. Each tuple contains an existing Sigma rule s_z, its relevant shell command c_z, and a corresponding attack description d_z, forming a set of foundational command-to-Sigma rule mappings.

Secondly, to enhance the model's robustness against command variations across different system versions and attack environments, we enrich the corpus. For each original command c_z, we generate $\mathbf{V}$ semantically equivalent variants $\{c_z^v\}_{v=1}^{\mathbf{V}}$ and M natural-language queries $\{q_z^m\}_{m=1}^{M}$, all paired with the same Sigma rule s_z. The augmented set is defined as:

$$R_z = \{(sp, c_z^v, q_z^m, s_z) \mid 1 \le v \le \mathbf{V}, 1 \le m \le M\}, \quad (9)$$

where sp is a fixed system prompt. This expansion ensures coverage of diverse command expressions.

Finally, to enable the model to internalize the mapping knowledge efficiently, we train a lightweight LoRA adapter $\Delta\theta_z = \{A_z, B_z\}$ for each command variant set. The feed-forward weight is decomposed as:

$$W' = W + AB^{\top}, \quad A \in \mathbb{R}^{h\times r}, B \in \mathbb{R}^{k\times r}, r \ll \min(h,k), \quad (10)$$

where $W \in \mathbb{R}^{h\times k}$ is the frozen pre-trained weight matrix. These parameters are optimized via the language modeling objective:

$$\min_{\Delta\theta_z} \sum_{(sp,c_z^v,q_z^m,s_z)\in R_z} \sum_{t=1}^{T} -\log P_{\theta+\Delta\theta_z}(r_t \mid r_{<t}), \quad (11)$$

with $r = sp \oplus c_z^v \oplus q_z^m \oplus s_z$. The optimized adapter parameters $\Delta\theta_z$ are then encoded into a compact representation $\mathbf{p}_z$. The resulting adapter representations $\mathbf{p}_z$ are stored in a parametric cache $\mathcal{M}$.

Sigma Rule Generation. To generate the final Sigma rule, we retrieve the top-k most similar adapter parameters $\{\Delta\theta_z\}$ from cache $\mathcal{M}$ for command $c_j \in \mathcal{C}^{(i)}$ and its verified description $d_{\text{desc}}^{(i)}$, where $\mathcal{N}_K(c_j)$ denotes the set of indices for the k-nearest neighbors of c_j in the cache. These parameters are aggregated and injected into $\mathcal{L}_{\text{base}}$:

$$\Delta W_{\text{merge}} = \frac{1}{K} \sum_{z\in\mathcal{N}_K(c_j)} A_z B_z^{\top}, \quad (12)$$

yielding the enhanced generator $\mathcal{L}_s = \mathcal{L}_{\text{base}} + \Delta W_{\text{merge}}$. The final Sigma rule s_{c_j} is generated by $\mathcal{L}_s$ using the prompt template in Appendix A.3, which incorporates both the command c_j and its security context $d_{\text{desc}}^{(i)}$.

5 Experiment Setup

Datasets and Models. We decompose **SIGMERGE** into seven core tasks, each supported by a dedicated dataset. Table 1 summarizes all datasets, including their name, sizes, split ratios, and sources. Since the raw sources must be adapted to

Table 1: Detailed overview of the datasets for the three modules of **SIGMERGE**. The **Notes** beneath the table provide detailed explanations of dataset construction and split strategies.

Module	Task	Dataset Name	Train	Dev	Test	Split Ratio	Sources	Validation Method
Information Extraction	❶ Tactic Extraction	ATT&CK	11,134	1,392	1,392	8:1:1	ATT&CK [3], Atomic Red Team [1]	Authoritative repositories
		Atomic Red Team	3,861	483	483	8:1:1		
	❷ Technique Extraction	ATT&CK	11,134	1,392	1,392	8:1:1	ATT&CK [3], Atomic Red Team [1]	Authoritative repositories
		Atomic Red Team	3,861	483	483	8:1:1		
	❸ Command Extraction	CommandX	8,715	-	2,179	8:2	Atomic Red Team [1], Metta [2], PowerShell [4]	Authoritative repositories
Attack Description Generation	❹ Knowledge Retrieval	External Knowledge	No training, total 613			-	CWE, CAPEC, ATT&CK (Mitigation, CoA, Procedure)	Authoritative repositories
	❺ Description Generation	AttackDesc-DPO	37,828	8,106	8,106	7:1.5:1.5	Based on External Knowledge	Manual inspection, code verification
	❻ Self-correction	AttackDesc-SC	28,339	6,073	6,073	7:1.5:1.5	Based on AttackDesc-DPO with injected random errors	Manual inspection
Sigma Rule Generation	❼ Sigma Rule Generation	ShellCom2Sigma	869	-	153	8.5:1.5	Sigma repository [57]	Validated on real SIEM systems

Notes. Following the recommendations of Arp et al. [7], we design our dataset partitions to mitigate common pitfalls such as *sampling bias*, *label inaccuracy*, *data snooping*, *spurious correlations*, and *biased parameter tuning*. For tactic and technique extraction, we follow prior ATT&CK-based practice and maintain consistent class proportions across training, validation, and test splits. This helps reduce sampling bias and ensures that each subset remains representative of the overall data distribution. For CommandX, our focus is on learning a robust command extractor rather than tuning task-specific hyperparameters. As a result, we do not introduce a separate development set and instead adopt a simple train–test split to keep the evaluation protocol straightforward and free from potential information leakage. In contrast, AttackDesc-DPO and AttackDesc-SC involve generative modeling, where a development set is necessary to adjust generation behavior without relying on test data. Given the larger dataset size, we allocate a dedicated validation split while preserving sufficient training coverage and a clear separation between training, tuning, and evaluation. Finally, for ShellCom2Sigma, which employs retrieval-augmented Sigma rule generation, we retain a relatively larger training set to support retrieval diversity. The test set is kept strictly disjoint, and the retrieval index is constructed solely from training data, ensuring that evaluation reflects genuine generalization rather than artifacts of data overlap.

SIGMERGE's required format for this new task, we also describe the validation procedures used to validate their correctness. For TTPs extraction, we curate two datasets—❶ tactics and ❷ techniques—from ATT&CK and Atomic Red Team sources. ❸ For command extraction, we build CommandX, and its test set is further divided into CommandX-1 (single or no command), CommandX-2 (two or no commands), and CommandX-M (multiple or zero commands) to evaluate robustness. ❹ The knowledge retrieval component uses a static knowledge base of 613 technique–knowledge mappings without training. For attack description generation, we introduce two datasets: ❺ AttackDesc-DPO for attack description generation, and ❻ AttackDesc-SC for self-correction. ❼ For Sigma rule generation, we construct ShellCom2Sigma to inject domain expertise into LLMs. For usability and reproducibility, we use a lightweight model. All LLM components in **SIGMERGE** are implemented with Qwen3-1.7B, which offers sufficient capability while keeping the computational cost low and the framework easy to deploy.

Baselines. We evaluate the TTPs extraction task against 16 representative baselines grouped into 4 categories:

(1) TTPs-specific description methods.

- AutoMap [33]: a DistilBERT-based pipeline with post-processing to map CTR text to TTPs.
- rcATT [32]: a Word2Vec-based method that infers TTPs from CTR descriptions.

(2) Pretrained and fine-tuned language methods.

Following the most recent research on TTPs extraction [10], which surveys the pretrained and fine-tuned models used in this task, we adopt the representative models identified in that study as our evaluation baselines:

- XLNet [68]: a generalized autoregressive pretraining method that learns bidirectional context.
- XLM-RoBERTa [40]: a replication study of BERT pretraining that analyzes data and hyperparameter effects to reveal overlooked design choices.
- BERT [15, 50, 60, 66]: a masked language model encoder designed to learn contextual representations for text classification.
- DistilBERT [52]: a compact language model trained with knowledge distillation during pretraining.
- CyBERT [49]: a BERT model adapted to cybersecurity through fine-tuning on CTR corpora.
- CTI-BERT [46, 70]: a BERT model trained on CTR corpora for cybersecurity text understanding.
- SciBERT [9, 48]: a BERT model pretrained on scientific publications for scientific NLP tasks.
- SecureBERT [6, 21, 51]: a BERT model tailored to cybersecurity text through domain-specific pretraining.

(3) Unsupervised methods.

- Raconteur [14]: uses LLMs with retrieval to align commands with TTPs, and it is important to note that the method operates on command text rather than descriptive text.

(4) General text classification models.

- CMAS [63]: a cooperative multi-agent framework that performs zero-shot classification through self annotation.
- GAProtoNet [64]: a graph attention prototypical network that models semantic relations for interpretable prediction.

- BiAttention [24]: enhances label–token alignment with label embeddings and bi-attention.
- BMRU [23]: combines a pre-trained encoder with recurrent and adaptive coupling units for cross-domain classification.

For the remaining tasks, we further include mainstream LLM families such as ChatGPT-5.1, Claude Sonnet 4.5, DeepSeek-V3 [36], DeepSeek-R1 [20], GLM4 [18], GLM4-9B [18], and Qwen3 [67]. To examine the impact of model architecture and parameter scale, we also compare the smaller Qwen3 models at 1.7B, 4B, and 8B. All models use their official default top-p and temperature settings and are evaluated without fine-tuning. For fairness, we set the few-shot prompt size to three for all tasks.

Implementation. We implement **SIGMERGE** in Python using PyTorch [47]. All hyperparameters are tuned on the validation set. For TTPs extraction, we train with Adam [31], using a batch size of 64 and a learning rate of 1e-5. The subsequence size k is tuned individually for each dataset. We apply early stopping with a patience of 20 epochs on the validation F1 score. For the remaining tasks, we fine-tune models using LlamaFactory, with learning rate 5e-5, 2 epochs, batch size 2, gradient accumulation 8, and a cosine learning rate scheduler. [3] All experiments are conducted on a single NVIDIA A6000 GPU (48GB) with CUDA 12.2.

Evaluation Metrics. We evaluate each module of **SIGMERGE** with task-specific metrics.

(1) Information extraction module.

For TTPs extraction, we report Micro-Precision (Mic-P), Micro-Recall (Mic-R), Micro-F1 score (Mic-F1), Label Ranking Average Precision (LRAP), and Hamming Loss (H-Loss), with micro-averaging to handle label imbalance. For command extraction, we use Precision (P), Recall (R), F1 score (F1) on CommandX test subsets (CommandX-1/2/M), and per-sentence inference time (Time).

(2) Attack description generation module.

We adopt standard text generation metrics:

- ROUGE-1/2/L (Recall-Oriented Understudy for Gisting Evaluation) [35]: Measures n-gram overlap between generated and reference descriptions.
- BLEU-1–4 (Bilingual Evaluation Understudy) [44]: Evaluates precision of n-gram matches with brevity penalty.
- METEOR (Metric for Evaluation of Translation with Explicit ORdering) [8]: Considers synonymy and stemming for improved correlation with human judgment.
- BERTScore [71]: Computes semantic similarity between generated and reference texts using contextual embeddings.

(3) Sigma rule generation module.

We design five categories of evaluation metrics:

- Structural and Value Matching (Stru&Val): Evaluates the correctness of rule structure and content matching. We focus on the `detection` field because it is the core component in Sigma rules that directly defines the detection logic. We formalize the `detection` field as a finite set of detection items $S = \{(k, V)\}$, where each key k is a field–operator pair and V is its associated value set. The computation details are defined as follows:

$$\text{Exact}(S^*, S) = \frac{|S^* \cap S|}{|S^* \cup S|}, \tag{13}$$

$$\text{Partial}(S^*, S) = \frac{1}{|K|} \sum_{k \in K} \text{Jaccard}\left(V(k), \hat{V}(k)\right). \tag{14}$$

- Keyword Matching: Value-only comparison with Macro-Precision (P), Macro-Recall (R), and Macro-F1 (F1) across rules.
- Condition Matching (C-Acc): Measures semantic equivalence of Boolean conditions after canonicalization.
- Composite Score (C-Score): The weights emphasize structural integrity (Exact: 0.3, Partial: 0.2) and logical correctness (C-Acc: 0.3), while also considering value matching accuracy (Macro-F1: 0.2). Since structural and logical fidelity are more critical than lexical matching in real-world Sigma rules, they are assigned higher weights than Keyword-F1.
- Efficiency (Time): The average end-to-end rule translation time.

6 Evaluation

This section presents a comprehensive evaluation of the **SIGMERGE** framework to demonstrate its effectiveness in automating the CTR-to-Sigma rule generation process. Experiments on information extraction are presented in §6.1, attack description generation in §6.2, and Sigma rule generation in §6.3. We further report a module-level ablation study in §6.4, a false positive and false negative analysis in §6.5, a case study on real-world CTRs in §6.6, and a comparative case study on **SIGMERGE** vs. Raconteur in §6.7.

6.1 Information Extraction Result

Result of the TTPs Extraction. As shown in Tables 2 and 3, **SIGMERGE** achieves the highest F1 scores across all baselines on both datasets for TTPs extraction. This superiority stems from three advantages: (1) **Compositional Semantics Modeling**: Unlike pretrained encoders that rely on lexical cues, our method distinguishes syntactically similar but functionally distinct commands; (2) **Domain-aware Structured Reasoning**: For CTR's fine-grained categories, LLM-based methods lack explicit reasoning, while our approach provides domain-tailored analysis; (3) **Hierarchical Representation**: General classifiers flatten features, whereas our model preserves tactic-technique hierarchies. Through multi-subsequence decomposition, **SIGMERGE** effectively captures the combinatorial

[3] https://github.com/hiyouga/LLaMA-Factory

Table 2: Main results of TTPs extraction for Tactic. The best results are in **bold**.

Model	ATT&CK						Atomic-Red-Team					
	Acc	Mic-P	Mic-R	Mic-F_1	LRAP	H-Loss(×1e-3)	Acc	Mic-P	Mic-R	Mic-F_1	LRAP	H-Loss(×1e-3)
rcATT	24.13	43.34	54.10	48.13	-	99.09	15.32	24.00	27.94	25.82	-	92.5
AutoMap	82.92	86.31	86.94	86.62	92.15	93.71	95.62	98.14	95.36	96.72	99.21	3.72
XLNet	85.63	87.28	87.82	87.55	93.82	17.27	96.66	97.40	96.90	97.15	98.78	3.27
XLM-RoBERTa	87.05	89.47	87.79	88.62	93.70	19.13	94.97	97.12	95.61	96.36	98.63	4.61
BERT	86.13	88.10	88.32	88.21	92.96	19.03	96.20	98.03	96.41	97.22	98.66	2.08
DistilBERT	86.38	90.09	87.67	88.86	18.65	22.84	95.08	98.01	95.04	96.50	98.50	3.99
CySecBERT	86.79	89.70	89.32	89.51	93.60	17.80	96.04	97.38	96.13	96.75	98.83	3.72
CyBERT	86.33	89.14	87.67	88.40	92.84	19.57	97.04	98.17	96.90	97.53	99.00	3.20
CTI-BERT	87.31	90.69	88.74	89.67	94.13	16.55	96.04	96.38	96.13	96.25	98.35	4.31
SciBERT	84.95	86.77	87.67	87.22	92.40	21.85	94.37	96.07	94.58	95.32	97.70	5.35
SecureBERT	87.22	90.03	89.81	89.92	93.78	17.13	96.67	97.17	97.42	97.29	98.73	3.12
Raconteur	30.43	38.16	32.10	34.87	52.40	101.83	20.53	20.53	25.34	22.68	52.72	3.01
CMAS	62.45	72.38	65.19	68.54	74.21	20.3	41.23	58.91	38.76	46.72	58.34	2.98
GAProtoNet	85.04	88.23	86.76	87.49	93.11	18.92	94.27	96.31	92.08	94.15	96.83	1.64
BiAttention	86.17	89.32	87.41	88.35	93.73	17.12	95.91	98.13	96.07	97.08	99.03	2.47
BMRU	86.43	89.71	87.89	88.79	93.56	17.44	96.12	98.27	96.23	97.23	99.08	2.18
SIGMERGE	**87.42**	**90.92**	**88.98**	**89.94**	**94.55**	**16.46**	**97.08**	**98.85**	**97.42**	**98.18**	**99.39**	**2.08**

Table 3: Main results of TTPs extraction for Technique. The best results are in **bold**.

Model	ATT&CK						Atomic-Red-Team					
	Acc	Mic-P	Mic-R	Mic-F_1	LRAP	H-Loss(×1e-3)	Acc	Mic-P	Mic-R	Mic-F_1	LRAP	H-Loss(×1e-3)
rcATT	5.17	20.40	17.83	19.03	-	3.68	6.00	23.49	8.07	12.01	-	1.80
AutoMap	62.64	87.32	67.24	75.97	81.81	1.03	49.16	**100**	49.16	65.92	93.42	0.77
XLNet	48.70	81.20	58.99	68.34	73.69	1.03	56.25	95.40	56.25	70.77	87.88	0.71
XLM-RoBERTa	23.06	82.31	22.35	35.16	64.35	68.40	14.73	97.05	14.73	25.58	39.14	1.30
BERT	48.85	87.92	57.10	69.23	76.10	1.08	6.47	96.66	6.47	12.13	23.91	1.43
DistilBERT	41.51	**92.60**	48.57	63.72	76.64	1.34	6.47	**100**	6.47	12.15	34.33	1.42
CySecBERT	67.87	85.27	72.39	78.31	82.60	1.97	14.37	**100**	14.37	25.13	62.86	33.27
CyBERT	50.07	79.61	54.68	64.83	78.65	49.26	15.20	**100**	15.20	26.40	60.16	34.05
CTI-BERT	64.75	83.50	69.31	75.75	86.82	36.85	42.50	99.51	42.50	59.56	81.29	0.87
SciBERT	67.36	81.47	72.85	76.92	81.67	20.07	71.87	98.85	71.87	83.23	95.73	0.44
SecureBERT	66.42	83.44	72.21	77.42	82.84	1.02	13.54	97.01	13.54	23.76	50.77	1.32
Raconteur	9.07	30.80	19.33	23.75	34.51	99.97	34.15	34.15	34.15	34.15	44.21	2.00
CMAS	8.12	35.72	16.09	22.18	39.61	28.00	35.14	40.72	36.09	38.28	39.61	1.86
GAProtoNet	51.74	80.16	66.83	72.88	80.17	1.19	79.43	80.16	67.01	73.02	80.17	1.96
BiAttention	58.98	82.89	71.17	76.60	79.94	1.12	80.65	82.89	71.17	76.60	89.93	0.94
BMRU	60.24	82.42	71.05	76.32	80.43	1.06	90.3	83.96	73.92	78.62	90.97	0.82
SIGMERGE	**69.62**	83.71	**75.27**	**79.27**	**83.52**	**0.99**	**97.92**	99.16	**97.92**	**98.53**	**98.35**	**0.18**

nature of attack behaviors, aligning with MITRE ATT&CK's framework.

Result of the Command Extraction. Table 4 shows that **SIGMERGE** achieves near-perfect F1 scores and consistently surpasses all 13 LLM baselines under identical few-shot prompting, with especially large gains on multi-command benchmarks. Three key observations emerge: (1) **Smaller models benefit from step-by-step reasoning**, as thinking modes help decompose complex inputs and localize command boundaries; (2) **Larger models often overgenerate**, since strong priors for explanatory narratives lead to fragmented or distorted commands; (3) **General-purpose LLMs lack syntactic grounding**, struggling with precise boundary detection due to limited pretraining on shell command syntax. In contrast, **SIGMERGE**, fine-tuned on CommandX, learns robust command structures and delivers high-precision, generalizable extraction, which provides a reliable foundation for downstream Sigma rule generation.

Ablation Study for the TTPs Extraction. We examine how the subsequence length k influences TTPs extraction in **SIGMERGE**. Figure 6 shows that performance is highly sensitive to k, with the best results occurring at $k = 3$ or $k = 4$ depending on dataset and task. Three insights emerge: (1) subsequence modeling captures relational patterns among attack elements, confirming the value of compositional decomposition; (2) $k = 3$ consistently works best for tactic classification across datasets, indicating that moderate context spans are sufficient for high-level stage recognition; (3) technique classification benefits from longer contexts in text-rich data (e.g., $k = 4$ on ATT&CK) but prefers shorter ones when commands are present (e.g., $k = 2$ on Atomic-Red-Team), since commands provide dense, localized cues that reduce the need for wide context.

Efficiency of Command Extraction. We compare the command extraction latency of 13 LLM variants from ChatGPT, Claude, Qwen, DeepSeek, and ChatGLM. Table 4 shows that **SIGMERGE** achieves the lowest inference time across all datasets, with over 2× speed-ups on average, while most

Table 4: Main comparison results on the CommandX-1, CommandX-2, and CommandX-M datasets. † indicates local deployment. ‡ indicates API call. The best results are in **bold**.

Model	CommandX-1					CommandX-2					CommandX-M				
	P	R	F1	Time(s)	Speed Up	P	R	F1	Time(s)	Speed Up	P	R	F1	Time(s)	Speed Up
ChatGPT5.1‡	93.72	96.14	94.29	2.02	0.32x	85.21	84.89	85.05	2.32	0.80x	88.60	88.63	88.62	2.19	0.69x
Claude Sonnet 4.5‡	88.45	92.45	90.41	4.79	0.14x	80.00	79.39	80.09	5.26	0.35x	86.41	86.41	86.41	5.93	0.25x
DeepSeek-V3‡	51.31	75.48	61.11	6.97	0.09x	67.36	82.74	74.27	8.96	0.21x	62.08	80.32	70.03	8.29	0.18x
DeepSeek-R1‡	80.99	84.36	82.64	66.88	0.01x	60.24	60.99	60.61	125.96	0.01x	67.38	68.53	67.95	104.06	0.01x
GLM4$_{(9B)}$†	13.86	40.25	20.61	14.50	0.04x	14.78	36.93	21.11	18.75	0.10x	15.64	38.30	22.02	17.89	0.08x
GLM4Plus‡	34.84	63.31	44.94	3.87	0.17x	51.48	72.42	60.18	4.39	0.42x	47.06	69.92	56.25	4.73	0.32x
Qwen3$_{(1.7B)}$†	22.92	39.58	29.03	0.65	1.00x	13.20	25.69	17.44	1.86	1.00x	16.45	30.65	21.41	1.51	1.00x
Qwen3$_{(4B)}$†	25.22	54.85	34.56	1.03	0.63x	26.96	29.40	28.13	1.99	0.93x	26.25	38.99	31.55	1.63	0.93x
Qwen3$_{(4B)\text{Think}}$†	58.09	72.96	64.68	16.99	0.04x	41.80	49.78	45.44	34.47	0.05x	46.61	57.61	51.62	35.64	0.04x
Qwen3$_{(8B)}$†	24.47	54.83	33.84	3.78	0.17x	20.88	37.19	26.75	3.38	0.55x	22.65	43.07	29.68	3.23	0.47x
Qwen3$_{(8B)\text{Think}}$†	58.30	72.23	64.52	20.16	0.03x	39.54	51.22	44.63	44.99	0.04x	45.71	58.11	51.17	32.17	0.05x
QwenPlus‡	77.77	85.76	81.57	1.63	0.40x	70.98	81.51	75.88	2.86	0.65x	73.46	82.93	77.91	2.45	0.62x
QwenPlus$_{\text{Think}}$‡	68.72	82.30	74.90	27.48	0.02x	63.46	72.10	67.51	62.95	0.03x	65.53	75.48	70.15	49.29	0.03x
SIGMORGE	**99.57**	**99.47**	**99.52**	**0.47**	**1.38x**	**99.30**	**99.36**	**99.33**	**0.87**	**2.14x**	**99.39**	**99.36**	**99.38**	**0.75**	**2.01x**

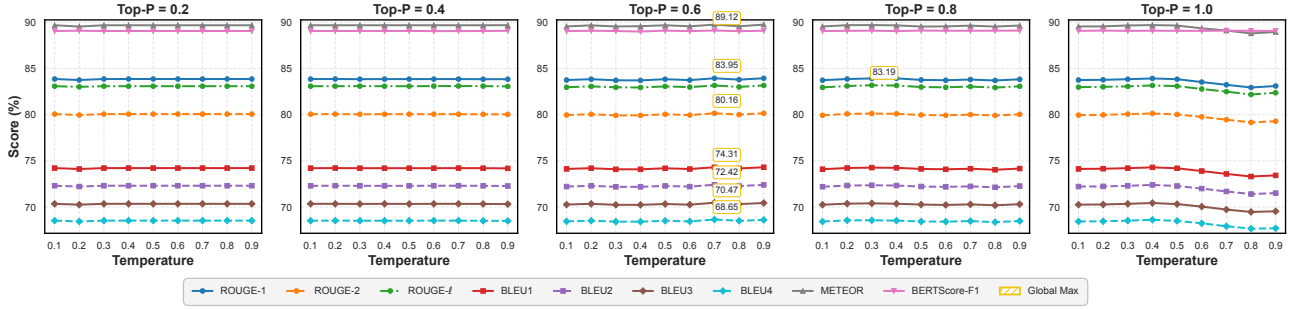

Figure 5: The impact of different top-p and temperature parameters on the results of attack description generation.

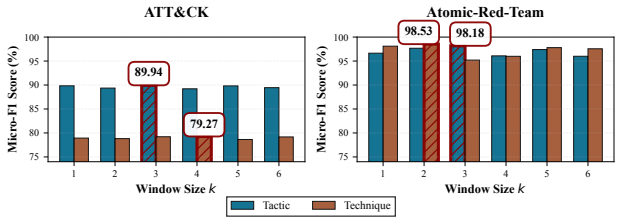

Figure 6: Comparison of window size k impact on TTPs extraction performance across two datasets.

baselines are substantially slower. Two insights emerge: (1) remote invocation (‡) consistently incurs high latency due to network and service overhead, even for relatively small models; (2) reasoning modes (e.g., Think) drastically increase latency, up to 60×, despite similar parameter counts. In contrast, **SIGMORGE**, as a lightweight fine-tuned model, delivers efficiency consistently across diverse datasets, showing that domain-specific specialization not only improves accuracy but also eliminates unnecessary overhead, making command extraction both faster and more practical.

6.2 Attack Description Generation Result

Result of Attack Description Generation. We evaluate **SIGMORGE** on generating natural language attack descriptions that conform to the standardized template **T**, which bridges high-level attack narratives with machine-level threat representations. As shown in Table 5, **SIGMORGE** achieves the highest scores across all metrics, surpassing strong baselines and consistently outperforming both smaller and larger variants. Three insights emerge: (1) **Template Fidelity**: **SIGMORGE** reliably fills every field in **T**, ensuring structural completeness and unifying heterogeneous threat reports into a standardized format; (2) **Context Grounding**: the *Background* field requires implicit extraction of attackers, targets, and tools from CTRs, and results confirm that **SIGMORGE** can faithfully recover such entities and embed them into coherent narratives; (3) **Scale Robustness**: while medium-scale baselines peak at balancing fluency and structure, larger models often generate fluent but structurally incomplete text, omitting fields or adding extraneous content, highlighting that scale without structural supervision leads to overgeneralization. In contrast, **SIGMORGE** fine-tuned with template preserving objectives delivers both natural and structurally faithful descriptions, providing reliable standardized narratives for downstream rule generation.

Ablation Study for the Attack Description Generation. We analyze the impact of top-p and temperature (τ) on textual attack description generation, as shown in Figure 5. Performance remains flat when $p \leq 0.4$ because the nucleus set collapses to high-probability tokens, yielding near-deterministic

Table 5: Performance of **SIGMERGE** on the AttackDesc-DPO test set. † indicates local deployment. ‡ indicates API call. The best results are in **bold**.

Model	ROUGE-1	ROUGE-2	ROUGE-ℓ	BLEU-1	BLEU-2	BLEU-3	BLEU-4	METEOR	BERTScore-P	BERTScore-R	BERTScore-F1
ChatGPT5.1‡	75.58	69.51	74.07	59.99	57.39	55.10	53.09	84.27	82.63	89.84	86.04
Claude Sonnet 4.5‡	65.99	61.06	64.44	49.79	47.79	46.05	44.53	78.89	80.27	90.23	84.90
DeepSeek-V3‡	67.71	64.38	67.02	51.49	50.12	48.89	47.75	80.11	80.82	**91.49**	85.76
GLM4$_{\text{(9B)}}$†	80.98	73.78	77.44	72.04	68.56	65.65	63.24	79.28	86.97	87.38	87.18
GLM4Plus‡	82.99	78.45	81.69	70.39	68.23	66.43	64.77	86.26	84.01	90.64	87.14
Qwen3$_{\text{(1.7B)}}$†	81.60	75.78	80.12	72.77	70.08	67.69	65.52	81.86	87.56	89.93	88.66
Qwen3$_{\text{(4B)}}$†	83.05	77.45	81.65	73.37	70.76	68.56	66.67	85.47	**88.33**	87.92	88.09
Qwen3$_{\text{(8B)}}$†	74.33	67.72	72.69	63.48	60.18	57.55	55.22	77.84	82.16	86.98	84.44
QwenPlus‡	70.75	66.18	69.46	55.78	53.84	52.17	50.65	80.83	81.04	89.77	85.10
SIGMERGE	**83.95**	**80.16**	**83.16**	**74.31**	**72.42**	**70.47**	**68.65**	**89.76**	87.31	91.17	**89.12**

Table 6: Performance of **SIGMERGE** on the AttackDesc-SC test set. The best results are in **bold**.

Condition	ROUGE-1	ROUGE-2	ROUGE-ℓ	BLEU-1	BLEU-2	BLEU-3	BLEU-4	METEOR	BERTScore-P	BERTScore-R	BERTScore-F1
Raw Data	42.36	36.74	40.85	34.27	31.15	28.83	26.91	52.48	69.85	73.12	71.42
SIGMERGE	**81.27**	**76.83**	**80.45**	**71.64**	**68.92**	**66.38**	**64.12**	**86.73**	**86.95**	**90.74**	**88.80**
Increased	+38.91	+40.09	+39.60	+37.37	+37.77	+37.55	+37.21	+34.25	+17.10	+17.62	+17.38

outputs that adhere to **T** but suppress lexical variation. Once p increases, controlled diversity emerges and metrics begin to diverge. Peak ROUGE-1/2 and BLEU-1–4 appear at $p = 0.6$ and $\tau = 0.7$, where moderate stochasticity explores frequent paraphrases while $\mathcal{G}$ and **T** keep semantics and structure grounded. ROUGE-ℓ peaks at $p = 0.8$ and $\tau = 0.3$ because a broader nucleus with a cooler sampler preserves content order and length, improving longest-subsequence matches. Across settings, structural compliance remains stable due to template constraints and template-preserving fine-tuning.

Error Correction Capability. SIGMERGE achieves substantial gains across ROUGE, BLEU, METEOR, and BERTScore while preserving template fidelity, confirming the effectiveness of self-correction, as shown in Table 6. These gains are attributed to: (1) **Field-aligned validation**: Validators for each template field verify content against the filtered knowledge base $\mathcal{G}$, removing inaccuracies and resolving semantic inconsistencies; (2) **Template-aware localization**: Corrections target only erroneous text spans, preserving valid content and improving overall output quality. Collectively, these mechanisms convert noisy drafts into structurally sound and semantically coherent attack descriptions, providing reliable input for Sigma rule generation.

Table 7: Performance on ShellCom2Sigma dataset for Sigma rule generation. † indicates local deployment. ‡ indicates API call. The best results are in **bold**.

	Model	Stru&Val		Keyword (Macro)			Condition	Composite	Efficiency
		Exact	Partial	P	R	F1	C-Acc	C-Score	Time(s)
None RAG	**ChatGPT5.1**‡	2.73	20.02	23.60	43.81	26.67	15.02	14.66	3.07
	Claude Sonnet 4.5‡	2.11	15.77	28.52	41.29	30.33	58.49	27.40	199.14
	DeepSeek-V3‡	6.77	15.84	34.02	33.65	29.28	84.16	36.30	5.80
	DeepSeek-R1‡	3.62	11.89	30.98	34.28	30.38	86.96	35.63	280.33
	GLM4$_{\text{(9B)}}$†	0.82	6.47	22.12	25.26	20.01	83.56	30.61	5.35
	GLM4Plus‡	4.69	13.15	28.68	32.77	26.25	83.99	34.48	3.87
	Qwen3$_{\text{(1.7B)}}$†	0.89	6.49	23.82	25.48	21.14	84.01	30.99	0.96
	Qwen3$_{\text{(4B)}}$†	1.41	8.57	22.95	23.39	19.62	84.09	31.29	1.85
	Qwen3$_{\text{(8B)}}$†	2.48	10.68	27.04	30.20	24.69	84.17	33.07	1.39
	QwenPlus‡	5.73	17.25	29.89	33.23	27.11	82.79	35.43	2.75
	QwenPlus$_{\text{Think}}$‡	3.02	9.72	27.12	30.68	24.82	84.17	33.07	50.48
Standard RAG (k-Shot)	**ChatGPT5.1**‡ (5)	4.40	17.61	30.28	39.11	29.77	35.79	21.53	3.06
	ChatGPT5.1‡ (10)	7.89	21.58	38.11	40.77	35.09	63.68	32.80	3.26
	ChatGPT5.1‡ (15)	6.77	17.77	31.45	39.68	31.38	50.29	26.95	3.39
	ChatGPT5.1‡ (20)	6.42	18.54	30.52	39.20	30.28	37.99	23.09	3.59
	Claude Sonnet 4.5‡ (5)	5.13	20.52	32.97	46.26	33.16	49.71	27.19	45.02
	Claude Sonnet 4.5‡ (10)	11.32	23.89	41.78	40.61	36.80	71.43	36.96	26.39
	Claude Sonnet 4.5‡ (15)	11.90	22.50	42.90	37.62	35.63	74.31	37.49	33.98
	Claude Sonnet 4.5‡ (20)	12.11	23.43	43.79	38.04	36.53	73.56	37.69	20.37
	DeepSeek-V3‡ (5)	7.38	12.14	49.70	43.46	38.66	60.00	30.37	6.85
	DeepSeek-V3‡ (10)	10.83	25.58	46.67	46.40	38.30	85.00	41.52	7.30
	DeepSeek-V3‡ (15)	11.67	21.91	56.88	48.60	44.87	75.00	39.36	7.65
	DeepSeek-V3‡ (20)	15.00	22.31	56.62	53.32	48.19	65.00	38.10	8.30
	QwenPlus‡ (5)	0.50	16.21	34.09	39.26	29.75	65.00	30.19	3.05
	QwenPlus‡ (10)	7.50	19.05	49.45	38.48	36.37	80.00	37.33	3.32
	QwenPlus‡ (15)	14.17	21.21	44.99	50.40	40.80	85.00	42.15	3.68
	QwenPlus‡ (20)	11.11	17.27	35.53	44.86	33.85	83.33	38.56	3.75
Ours	**SIGMERGE**	**60.85**	**70.46**	**76.60**	**75.00**	**75.78**	**90.65**	**74.69**	**2.56**

6.3 Sigma Rule Generation Result

Result of Sigma Rule Generation. On the *ShellCom2Sigma* dataset, **SIGMERGE** surpasses both *None-RAG* and *Standard-RAG*, achieving the highest C-Score while maintaining practical efficiency, as shown in Table 7. The gains come from three key advantages. **First**, by internalizing retrieved knowledge into parameters rather than appending it to the context, **SIGMERGE** encodes stable patterns of Sigma syntax, log fields, and attack behaviors, leading to more faithful structural reconstruction. **Second**, parametric fusion provides persistent access to relevant values without competing for limited attention span, enabling more precise command-to-field mappings and reducing spurious matches. **Finally**, this design reduces the need for lengthy exemplars and iterative refinements, delivering both higher rule accuracy and faster convergence compared to context-only RAG. These benefits explain the consistent improvements across structure, value, and condition metrics, resulting in a stronger overall C-Score.

Table 8: Ablation study on contributions of Attack Description Generation and Sigma Rule Generation under Standard-RAG and ParamCache-RAG settings. Best results in **bold**. None indicates that no external knowledge is used.

Contribution	Model (k-Shot)	Stru&Val		Keyword (Macro)			Condition	Composite
		Exact	Partial	P	R	F1	C-Acc	C-Score
Attack Description Generation Module Contribution (None-RAG)	SIGMERGE(0) + $\mathcal{K}$[*None*] w/o DPO w/o SC	0.94	6.39	23.09	26.01	21.02	79.85	29.72
	SIGMERGE(0) + $\mathcal{K}$[*CWE*] w/o DPO w/o SC	2.31	11.52	26.87	22.41	24.42	80.12	31.92
	SIGMERGE(0) + $\mathcal{K}$[*CAPEC*] w/o DPO w/o SC	2.47	12.10	27.53	24.63	26.03	78.64	31.96
	SIGMERGE(0) + $\mathcal{K}$[*Mitigation*] w/o DPO w/o SC	4.22	13.02	31.41	28.19	29.47	82.42	34.49
	SIGMERGE(0) + $\mathcal{K}$[*CoA*] w/o DPO w/o SC	5.12	14.55	36.73	35.02	35.82	86.60	37.59
	SIGMERGE(0) + $\mathcal{K}$[*CWE*+*CAPEC*+*Mitigation*+*CoA*] w/o DPO w/o SC	5.80	15.10	42.88	41.33	41.66	84.93	38.57
	SIGMERGE(0) + $\mathcal{K}$[*CWE*+CAPEC+*Mitigation*+*CoA*] + DPO w/o SC	6.75	19.94	48.28	45.63	46.19	84.09	40.48
	SIGMERGE(0) + $\mathcal{K}$[*CWE*+*CAPEC*+*Mitigation*+*CoA*] + DPO + SC	6.83	21.12	51.77	48.97	50.05	83.24	41.25
Sigma Rule Generation Module Contribution (Standard-RAG)	SIGMERGE(5) + $\mathcal{K}$[*CWE*+*CAPEC*+*Mitigation*+*CoA*] + DPO + SC	4.63	15.92	38.11	40.25	38.92	82.51	37.11
	SIGMERGE(10) + $\mathcal{K}$[*CWE*+*CAPEC*+*Mitigation*+*CoA*] + DPO + SC	9.87	21.73	46.34	48.02	48.85	87.11	43.21
	SIGMERGE(15) + $\mathcal{K}$[*CWE*+*CAPEC*+*Mitigation*+*CoA*] + DPO + SC	11.13	24.52	49.89	51.41	52.37	87.81	45.06
	SIGMERGE(20) + $\mathcal{K}$[*CWE*+*CAPEC*+*Mitigation*+*CoA*] + DPO + SC	10.19	22.80	45.52	48.33	48.10	86.28	43.12
	SIGMERGE$_{\text{Warm}}$(20) + $\mathcal{K}$[*CWE*+*CAPEC*+*Mitigation*+*CoA*] + DPO + SC	26.48	36.39	68.20	68.84	68.52	87.03	55.03
Sigma Rule Generation Module Contribution (ParamCache-RAG)	SIGMERGE(5) + $\mathcal{K}$[*CWE*+*CAPEC*+*Mitigation*+*CoA*] + DPO + SC	11.26	23.02	52.82	50.69	51.26	86.11	44.07
	SIGMERGE(10) + $\mathcal{K}$[*CWE*+*CAPEC*+*Mitigation*+*CoA*] + DPO + SC	17.28	27.57	61.19	64.37	62.71	88.04	49.65
	SIGMERGE(15) + $\mathcal{K}$[*CWE*+*CAPEC*+*Mitigation*+*CoA*] + DPO + SC	28.33	33.21	68.68	67.58	68.10	87.21	54.92
	SIGMERGE(20) + $\mathcal{K}$[*CWE*+*CAPEC*+*Mitigation*+*CoA*] + DPO + SC	35.94	47.27	75.42	73.87	74.61	89.14	61.90
	SIGMERGE$_{\text{Warm}}$(20) + $\mathcal{K}$[*CWE*+*CAPEC*+*Mitigation*+*CoA*] + DPO + SC	**60.85**	**70.46**	**76.60**	**75.00**	**75.78**	**90.65**	**74.69**

6.4 Module-Level Ablation Study

To understand how each module contributes to resolving the *semantic gap* and the *domain-specific constraints*, we conduct a module-level ablation study in **SIGMERGE**. Table 8 examines two questions: (1) how external knowledge used for attack description generation influences the quality of the final Sigma rules, and (2) how different prompting strategies for Sigma rule generation behave under three settings, namely None-RAG, Standard-RAG, and ParamCache-RAG.

Contribution of Attack Description Module Generation. We first disable retrieval, namely None-RAG, and generate Sigma rules solely from attack descriptions produced with external knowledge. As shown in Table 8, among the four knowledge types, using *CoA* yields the largest gain, increasing the C-Score by 7.78. *CoA* provides explicit detection oriented guidance, which helps the model map attack semantics to concrete detection content and cover malicious parameter variants in commands. In contrast, using *CWE* yields the smallest gain, improving the C-Score by 2.20. This is expected because not every attack step can be aligned with a *CWE* entry, and *CWE* descriptions are not directly tied to detection behavior, although they still provide coarse semantic guidance. Incorporating DPO and self-correction further improves performance, indicating that template constrained refinement enhances semantic precision and structural consistency. Overall, this module mainly addresses the semantic gap caused by command variability and diverse malicious arguments. This effect is best reflected by Macro-F1, since it focuses on matching the detection values associated with operators rather than the operators themselves. Accordingly, moving from no external knowledge to the full set of *CoA*, *CWE*, *CAPEC*, and *Mitigation* increases Macro-F1 by 29.03.

Contribution of Sigma Rule Module Generation. Next, we examine how prompting strategies affect Sigma rule construction. Under Standard-RAG, performance improves relative to None-RAG, with the C-Score rising from 41.25 to 45.06 at 15 shots. However, a performance dip is observed at 5 shots, suggesting that a small number of demonstrations can cause the model to over rely on specific prompt examples and produce incorrect rules instead of learning generalizable detection logic. In contrast, ParamCache-RAG shows a consistent performance gain as the number of shots increases, reaching a C-Score improvement of 20.65 over None-RAG at 20 shots. Compared with Standard-RAG, this result indicates that injecting knowledge through model parameters is more effective than relying on context based prompting, as it allows the model to internalize domain-specific patterns rather than imitate surface level examples. Incorporating warm-up fine tuning further strengthens this effect, improving structural correctness by 44.97. Overall, this module primarily enhances structural accuracy. From 0 shot to 20 shots under ParamCache-RAG, the Exact score increases by 29.11, which contributes most to the overall C-Score due to its emphasis on rule level correctness.

Table 9: False-negative and false-positive analysis of generated Sigma rules.

Test Type	Samples	Correct	Error Rate	Accuracy
Positive (FNR)	200	176	12.0%	88.0%
Negative (FPR)	600	557	7.17%	92.83%

Key Findings. Attack Description Generation mitigates the semantic gap by combining external knowledge with template guided descriptions, enabling the LLM to capture diverse malicious parameter variants in commands. In contrast, Sigma Rule Generation enforces structural fidelity under domain-specific constraints by ensuring the correct use of detection operators and their associated fields and values. The latter therefore contributes more prominently to final rule quality,

Table 10: Case study on ten real-world CTR reports covering six MITRE ATT&CK tactics.

Case ID	CTR Type	Command Executable	Detection Success	Rule Validity	Information Extraction Time(s)	Attack Description Generation Time(s)	Sigma Rule Generation Time(s)	Total Time(s)	TTP Coverage
1	Defender GUI removal	✓	✓	✓	4.74	6.34	2.63	13.71	✓
2	Gootloader SEO poisoning	✓	✗	✓	5.35	4.98	2.10	12.43	✓
3	Defender scan deletion	✓	✓	✓	5.79	6.22	2.27	14.28	✓
4	Automated discovery cmds	✓	✓	✓	7.65	5.26	3.17	16.08	✓
5	Exchange ProxyShell exploit	✗	✗	✓	5.47	12.04	2.09	19.60	✓
6	XMRig cryptomining	✓	✓	✓	8.61	5.36	2.45	16.42	✓
7	QBot scheduled task	✓	✓	✓	7.53	7.51	2.36	17.40	✓
8	MSIExec impersonation	✓	✓	✓	5.69	4.12	3.79	13.60	✓
9	Network discovery loops	✓	✓	✓	11.20	4.26	3.16	18.62	✓
10	Remote process enumeration	✓	✓	✓	3.46	5.89	2.68	12.03	✓

since the correctness of operators, fields, and conditions is decisive for executable Sigma rules. Together, the two modules enable **SIGMERGE** to generate high-quality Sigma rules.

6.5 False Positive and False Negative Analysis

Using the ten case studies in §6.6 with ground-truth Sigma rules, we conduct a controlled false-negative and false-positive evaluation with executable command-level samples, as shown in Table 9. We generate 200 malicious samples as real-world command variants that are semantically equivalent to the attacks covered by the ground-truth rules but differ in syntax or parameters. A sample is counted as a false negative if it is not detected by the Sigma rule generated by **SIGMERGE**. In addition, we generate 600 benign samples that resemble malicious patterns but are non-malicious according to the ground-truth rules to measure false positives. All samples are executed in virtual machines and validated through a SIEM platform. Across 21,236 collected system logs, the generated rules yield a false-negative rate of 12.0% and a false-positive rate of 7.17%.

6.6 Case Study on Real-World CTRs

Experiment Setup. We evaluated the practical utility of **SIGMERGE** through a case study on ten CTRs from the DFIR platform. These reports were reproduced, generating 4,559 log events that spanned six tactics: privilege escalation, initial access, persistence, discovery, execution, and defense evasion. Each case was executed in a Windows 10 virtual machine instrumented with the QingTeng Agent [5], with process events collected and exported to Elastic SIEM for validation. To ensure full reproducibility, we will release all case details, including report sources, input fragments, system logs, intermediate steps, and final KQL queries.

Metrics. We assessed five dimensions aligned with Table 10: (1) *Command Executable*, whether the reported shell command was reproducible; (2) *Detection Success*, whether the generated Sigma rule could be compiled into KQL and successfully detect the attack; (3) *Rule Validity*, syntactic correctness of the generated rule; (4) *Time*, runtime of each module and the total pipeline; and (5) *TTP Coverage*, whether identified techniques aligned with the report's ATT&CK timeline.

Results and Key Findings. Table 10 summarizes the outcomes across ten CTR cases. Nine of ten reported commands were reproducible, confirming the realism of CTR-derived artifacts. Detection succeeded in eight cases, with failures only in Case 2 (Gootloader) due to overly strict rule conditions and Case 5 (ProxyShell) because of its environment-specific dependency. All generated Sigma rules were syntactically valid and thus machine-executable. The end-to-end runtime averaged only a few seconds per module, demonstrating computational efficiency suitable for near real-time deployment. Moreover, all identified TTPs matched the ground-truth ATT&CK coverage of the DFIR reports, validating the semantic completeness of the framework. Overall, these results demonstrate that **SIGMERGE**: (1) guarantees correctness through valid and executable rules; (2) achieves semantic completeness by aligning with ground-truth TTPs; (3) remains computationally efficient for practical deployment; and (4) provides diagnostic insights into environment-specific and overly strict conditions, offering directions for further enhancement.

6.7 Case Study on SIGMERGE vs. Raconteur

To examine how different forms of command interpretation influence Sigma rule generation, we compare **SIGMERGE** with Raconteur [14]. Both systems analyze attacker commands, but their goals differ: Raconteur assists analysts in understanding command behavior, whereas **SIGMERGE** converts CTRs into Sigma rules for automated defense. Because Raconteur does not release its datasets or training pipeline, our evaluation uses the five examples available on its project site.[4] We apply both systems to these examples and present two representative cases in Figure 7. The remaining cases are provided on our Zenodo repository.

Limitation. This comparison has two key limitations. First, the evaluation is restricted to the publicly released examples, precluding broader quantitative analysis. Second, the two systems are designed for distinct tasks, thus the goal of the

[4] https://raconteur-ndss.github.io/

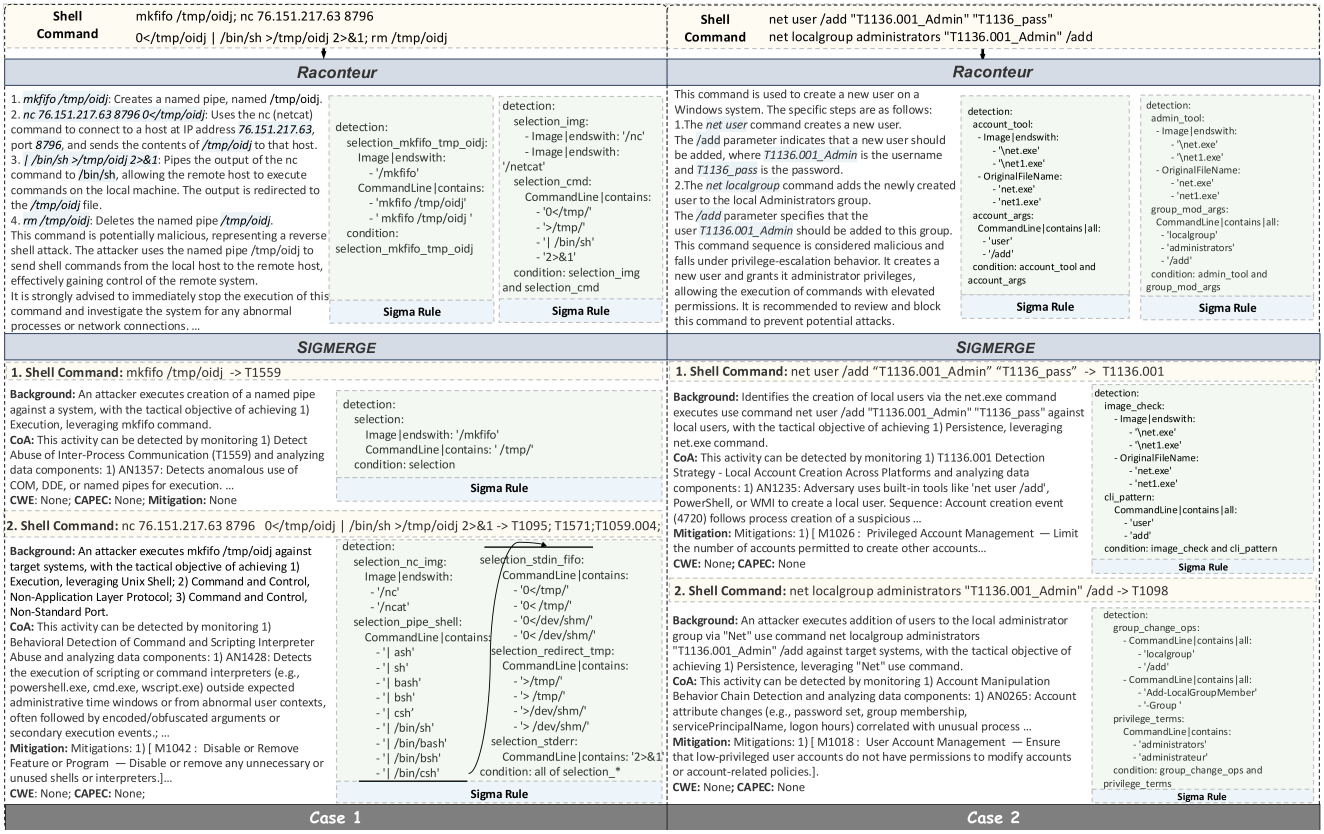

Figure 7: Comparing how **SIGMERGE**'s attack descriptions and Raconteur's command explanations influence downstream Sigma rule generation, with intermediate outputs shown.

comparison is not to rank their performance but to investigate how their interpretation styles influence downstream Sigma rule generation.

Case 1. Raconteur gives a detailed interpretation of the full command sequence. **SIGMERGE** operates differently because it processes one command at a time. The example contains three commands. All three exhibit malicious intent, but the main behaviors occur in the first two, so our analysis focuses on (1) creating a named pipe with `mkfifo /tmp/oidj` and (2) launching a remote shell through `nc 76.151.217.63 8796 0</tmp/oidj | /bin/sh >/tmp/oidj 2>&1`. For (1), both systems generate Sigma rules that detect FIFO creation. Raconteur matches the exact FIFO path, tying its rule to the specific file name. **SIGMERGE** generalizes to common temporary directories such as `/tmp/`, reducing sensitivity to attacker-chosen names. For (2), Raconteur identifies variants of netcat (e.g., `nc`, `netcat`), showing that LLM reasoning can resolve simple semantic differences and supporting the feasibility of command-to-Sigma translation. **SIGMERGE**, aided by ParamCache-RAG and external knowledge, expands the rule to include additional shell interpreters and common temporary file paths. This broader design results from combining description generation with retrieval.

Case 2. For `net user /add`, both systems produce correct rules and use appropriate operators while recognizing common variants such as `net` and `net1`. For the second command that modifies user privileges, Raconteur focuses on the syntax that appears in the input. **SIGMERGE** extends the rule to include PowerShell forms such as `Add-LocalGroupMember`, enabling detection of behavior that is equivalent to the given command rather than only its surface text.

Summary. Raconteur generates correct rules for the input commands and shows that LLMs can map command behavior to Sigma patterns. **SIGMERGE**, in comparison, is designed for automated detection and therefore emphasizes behavior generalization. It uses contextual information, external knowledge, and retrieval support to cover broader malicious variants.

7 Related Work

CTR analysis and outputs. Early work on CTRs primarily applied classical natural language processing (NLP) techniques such as entity recognition, relation extraction, and knowledge graph construction. Extractor [54], MF [39], SecIE [45], and cyNer [16] focused on extracting entities such

as malware, files, and processes from unstructured CTR text. Building on entities, Jia et al. [29] constructed attack graphs to represent attack paths, while TRACE [61] linked ATT&CK procedures to CWE and CoA to capture causal dependencies. Open CyKG [53] combined relation and entity extraction to produce open-domain threat graphs. To extend beyond entities, AttacKG [34] aligned CTR IoCs with ATT&CK techniques via graph reasoning, and TTPDrill [27] combined weak supervision and rules for TTPs extraction. Li et al. [33] leveraged DistilBERT with post-processing to map CTR sentences to ATT&CK labels. These approaches illustrate the feasibility of CTR mining, but they rely on pre-defined schemas and lack standardized datasets, limiting generalization. More critically, their outputs, entities, graphs, or TTPs, remain descriptive and human-facing, without producing machine-level artifacts for automated defense. With the advent of LLMs, CTR analysis shifted toward more flexible tasks. AttackG+ [73] used prompt engineering to jointly extract entities, relations, and TTPs, generating multi-dimensional attack graphs. ESP [11] adopted RAG to output relations in the structured threat information expression (STIX) format for structured intelligence sharing. Hu et al. [25] fine-tuned LLaMA2-7B to induce cyber knowledge graphs, while AECR [12] specialized in TTP recognition. Shafee et al. [56] benchmarked LLMs on CTR tasks such as binary classification, showing improved scalability. Although LLM-based methods demonstrate stronger generalization, they too stop at descriptive outputs, offering structured knowledge but not machine-level executable rules that can be directly integrated into detection pipelines.

Datasets for CTR tasks. Existing CTR datasets are largely task-specific, focusing on NER, relation extraction, or TTP classification. Typical sources include Symantec, Microsoft, DNRTI [62], and MalwareTextDB [11, 27, 39, 53], while other efforts rely on custom crawlers [16, 25, 37] or semi-manual LLM-assisted annotation [38, 73]. For rule construction, Cai et al. [11] annotated 198 CTRs in STIX [43] format to build Yara rules, but these rules are bound to code-level or runtime malware features that demand dynamic analysis and limit robustness. Similarly, Schwartz et al. [55] curated 20 AWS incident reports to derive Sigma rules from network IoCs (e.g., IPs, domains).

8 Conclusion

This paper presents **SIGMERGE**, the first end-to-end framework that automatically transforms high-level CTR text into machine-level Sigma rules without human intervention. By introducing a novel semantic descent paradigm, **SIGMERGE** bridges the gap between unstructured intelligence and operational defense. Extensive evaluation across 23 metrics and 10 real-world case studies demonstrates that each module is both effective and efficient, outperforming baselines while producing rules that integrate seamlessly with SIEM platforms. The framework remains efficient and lightweight, enabling rapid CTR analysis and deployment in real-world environments. Beyond evaluation, **SIGMERGE** has contributed four new Sigma rules accepted into the official repository and released seven domain-specific datasets, providing valuable resources for future research in CTR-driven detection.

In future work, we plan to extend CTR-driven rule generation beyond system log behaviors to broader domains such as vehicle forensics and malicious file analysis. We will also continue to refine and expand the CTR-domain datasets introduced in this work, and release additional code and resources to facilitate reproducibility and community adoption.

Acknowledgments

We thank the anonymous reviewers for their valuable comments. This work was supported in part by the National Natural Science Foundation of China (U24A20336, U24B20148, 62272114), National Science and Technology Major Project (2022ZD0119602), Major Key Project of PCL (PCL2024A05), the Science and Technology Program of Guangzhou (2024A03J0399), and Joint Research Fund of Guangzhou University (202201020380).

Ethical Considerations

In developing **SIGMERGE**, we encountered several ethical challenges. This section summarizes our key considerations and safeguards.

Stakeholders. Our analysis considers three groups: (1) security analysts who produce and rely on CTRs, (2) organizations that operate SIEM-based detection pipelines, and (3) the research community studying threat-intelligence automation and security-oriented LLM systems.

Benefits and Harms. For defenders, **SIGMERGE** can reduce manual workload, improve rule consistency, and accelerate the deployment of SIEM detections. These benefits help operational teams respond more effectively to real incidents. At the same time, we acknowledge potential risks. Automated rule generation may introduce inaccuracies or overly broad patterns if used without human review, and excessive reliance on automation could affect the quality of existing detection pipelines. The system does not process or generate exploit code, but its outputs still require professional validation before operational use.

Mitigations. To address these risks, we (1) design **SIGMERGE** to operate only on analyst-authored CTRs rather than raw system logs, (2) require that all generated rules undergo human inspection before deployment, and (3) avoid releasing sensitive operational data, limiting public artifacts to sources already available in open threat-intelligence repositories. These measures reduce the chance of misuse and help maintain the integrity of real-world detection workflows.

These safeguards reflect our intent to maximize defensive benefit while keeping residual risks low and well-bounded.

Rationale for Publication. We chose to publish this work because it advances automated cyber defense, aligns with community interest in responsible LLM use for security, and provides practical, measurable improvements for SOC workflows. These societal and operational benefits outweigh the minimal risk of misuse, given the system's dependence on defender-generated CTRs and its focus on detection logic rather than offensive capability.

Open Science

This work aligns with the principles of Open Science and aims to facilitate transparency, reproducibility, and community collaboration. All resources are available via our project repository at: https://doi.org/10.5281/zenodo.17970580. We provide full documentation and guidelines to replicate our experiments.

References

[1] Atomic red team. https://github.com/redcanaryco/atomic-red-team.

[2] Metta. https://github.com/uber-common/metta.

[3] Mitre att&ck. https://attack.mitre.org/.

[4] Powershell. https://github.com/PowerShell/PowerShell.

[5] Qingteng. https://www.qingteng.cn.

[6] Ehsan Aghaei, Xi Niu, Waseem Shadid, and Ehab Al-Shaer. Securebert: A domain-specific language model for cybersecurity. In *International Conference on Security and Privacy in Communication Systems*, pages 39–56. Springer, 2022.

[7] Daniel Arp, Erwin Quiring, Feargus Pendlebury, Alexander Warnecke, Fabio Pierazzi, Christian Wressnegger, Lorenzo Cavallaro, and Konrad Rieck. Dos and don'ts of machine learning in computer security. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 3971–3988, 2022.

[8] Satanjeev Banerjee and Alon Lavie. Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. In *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, pages 65–72, 2005.

[9] Iz Beltagy, Kyle Lo, and Arman Cohan. Scibert: A pretrained language model for scientific text. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3615–3620, 2019.

[10] Marvin Büchel, Tommaso Paladini, Stefano Longari, Michele Carminati, Stefano Zanero, Hodaya Binyamini, Gal Engelberg, Dan Klein, Giancarlo Guizzardi, Marco Caselli, et al. {SoK}: Automated {TTP} extraction from {CTI} reports–are we there yet? In *34th USENIX Security Symposium (USENIX Security 25)*, pages 4621–4641, 2025.

[11] Yongxin Cai, Jing Qiu, Fan Zhang, Qiang Li, and Lei Chen. A knowledge extraction framework on cyber threat reports with enhanced security profiles. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 326–336, 2025.

[12] Minghao Chen, Kaijie Zhu, Bin Lu, Ding Li, Qingjun Yuan, and Yuefei Zhu. Aecr: Automatic attack technique intelligence extraction based on fine-tuned large language model. *Computers & Security*, 150:104213, 2025.

[13] Wenrui Cheng, Tiantian Zhu, Tieming Chen, Qixuan Yuan, Jie Ying, Hongmei Li, Chunlin Xiong, Mingda Li, Mingqi Lv, and Yan Chen. Crucialg: Reconstruct integrated attack scenario graphs by cyber threat intelligence reports. *IEEE Transactions on Dependable and Secure Computing*, 2025.

[14] Jiangyi Deng, Xinfeng Li, Yanjiao Chen, Yijie Bai, Haiqin Weng, Yan Liu, Tao Wei, and Wenyuan Xu. Raconteur: A knowledgeable, insightful, and portable llm-powered shell command explainer. In *NDSS*, 2025.

[15] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pages 4171–4186, 2019.

[16] Shota Fujii, Nobutaka Kawaguchi, Tomohiro Shigemoto, and Toshihiro Yamauchi. Cyner: Information extraction from unstructured text of cti sources with noncontextual iocs. In *International workshop on security*, pages 85–104. Springer, 2022.

[17] Peng Gao, Xiaoyuan Liu, Edward Choi, Bhavna Soman, Chinmaya Mishra, Kate Farris, and Dawn Song. A system for automated open-source threat intelligence gathering and management. In *Proceedings of the 2021 International conference on management of data*, pages 2716–2720, 2021.

[18] Team GLM, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Dan Zhang, Diego Rojas, Guanyu Feng, Hanlin Zhao, et al. Chatglm: A family of large language models from glm-130b to glm-4 all tools. *arXiv preprint arXiv:2406.12793*, 2024.

[19] GreyNoise. Surge in threat actor exploitation attempts. https://gbhackers.com/surge-in-threat-actor-exploitation-attempts/, 2025.

[20] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[21] Zhiqiang Hao, Chuanyi Li, Xiao Fu, Bin Luo, and Xiaojiang Du. Leveraging hierarchies: Hmcat for efficiently mapping cti to attack techniques. In *European Symposium on Research in Computer Security*, pages 65–85. Springer, 2024.

[22] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.

[23] Jinpeng Hu, DanDan Guo, Yang Liu, Zhuo Li, Zhihong Chen, Xiang Wan, and Tsung-Hui Chang. A simple yet effective subsequence-enhanced approach for cross-domain ner. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 12890–12898, 2023.

[24] Jinpeng Hu, He Zhao, Dan Guo, Xiang Wan, and Tsung-Hui Chang. A label-aware autoregressive framework for cross-domain ner. In *Findings of the Association for Computational Linguistics: NAACL 2022*, pages 2222–2232, 2022.

[25] Yuelin Hu, Futai Zou, Jiajia Han, Xin Sun, and Yilei Wang. Llm-tikg: Threat intelligence knowledge graph construction utilizing large language model. *Computers & Security*, 145:103999, 2024.

[26] Yi-Ting Huang, R Vaitheeshwari, Meng-Chang Chen, Ying-Dar Lin, Ren-Hung Hwang, Po-Ching Lin, Yuan-Cheng Lai, Eric Hsiao-Kuang Wu, Chung-Hsuan Chen, Zi-Jie Liao, et al. Mitretrieval: Retrieving mitre techniques from unstructured threat reports by fusion of deep learning and ontology. *IEEE Transactions on Network and Service Management*, 21(4):4871–4887, 2024.

[27] Ghaith Husari, Ehab Al-Shaer, Mohiuddin Ahmed, Bill Chu, and Xi Niu. Ttpdrill: Automatic and accurate extraction of threat actions from unstructured text of cti sources. In *Proceedings of the 33rd annual computer security applications conference*, pages 103–115, 2017.

[28] Soyeong Jeong, Jinheon Baek, Sukmin Cho, Sung Ju Hwang, and Jong C Park. Adaptive-rag: Learning to adapt retrieval-augmented large language models through question complexity. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 7029–7043, 2024.

[29] Junbo Jia, Li Yang, Yuchen Wang, and Anyuan Sang. Hyper attack graph: Constructing a hypergraph for cyber threat intelligence analysis. *Computers & Security*, 149:104194, 2025.

[30] Beomjin Jin, Eunsoo Kim, Hyunwoo Lee, Elisa Bertino, Doowon Kim, and Hyoungshick Kim. Sharing cyber threat intelligence: Does it really help? In *Proceedings of the 31st Annual Network and Distributed System Security Symposium (NDSS)*, 2024.

[31] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[32] Valentine Legoy, Marco Caselli, Christin Seifert, and Andreas Peter. Automated retrieval of att&ck tactics and techniques for cyber threat reports. *arXiv preprint arXiv:2004.14322*, 2020.

[33] Lingzi Li, Cheng Huang, and Junren Chen. Automated discovery and mapping att&ck tactics and techniques for unstructured cyber threat intelligence. *Computers & Security*, 140:103815, 2024.

[34] Zhenyuan Li, Jun Zeng, Yan Chen, and Zhenkai Liang. Attackg: Constructing technique knowledge graph from cyber threat intelligence reports. In *European Symposium on Research in Computer Security*, pages 589–609. Springer, 2022.

[35] Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. In *Association for Computational Linguistics*, pages 74–81, 2004.

[36] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

[37] Jian Liu, Junjie Yan, Jun Jiang, Yitong He, Xuren Wang, Zhengwei Jiang, Peian Yang, and Ning Li. Tricti: an actionable cyber threat intelligence discovery system via trigger-enhanced neural network. *Cybersecurity*, 5(1):8, 2022.

[38] Jiehui Liu and Jieyu Zhan. Constructing knowledge graph from cyber threat intelligence using large language model. In *2023 IEEE International Conference on Big Data (BigData)*, pages 516–521. IEEE, 2023.

[39] Peipei Liu, Hong Li, Zuoguang Wang, Jie Liu, Yimo Ren, and Hongsong Zhu. Multi-features based semantic augmentation networks for named entity recognition in threat intelligence. In *2022 26th International Conference on Pattern Recognition (ICPR)*, pages 1557–1563. IEEE, 2022.

[40] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[41] Inoussa Mouiche and Sherif Saad. Entity and relation extractions for threat intelligence knowledge graphs. *Computers & Security*, 148:104120, 2025.

[42] Dongliang Mu, Alejandro Cuevas, Limin Yang, Hang Hu, Xinyu Xing, Bing Mao, and Gang Wang. Understanding the reproducibility of crowd-reported security vulnerabilities. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 919–936, 2018.

[43] OASIS Cyber Threat Intelligence (CTI) Technical Committee. Structured threat information expression (stix) version 2.1. https://docs.oasis-open.org/cti/stix/v2.1/os/stix-v2.1-os.html, 2025. Standardized XML/JSON serialization format for cyber threat intelligence exchange.

[44] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318, 2002.

[45] Youngja Park and Taesung Lee. Full-stack information extraction system for cybersecurity intelligence. In *Proceedings of the 2022 conference on empirical methods in natural language processing: industry track*, pages 531–539, 2022.

[46] Youngja Park and Weiqiu You. A pretrained language model for cyber threat intelligence. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 113–122, 2023.

[47] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Proceedings of the 33rd Annual Conference on Neural Information Processing Systems (NeurIPS)*, 32, 2019.

[48] Fariha Ishrat Rahman, Sadaf Md Halim, Anoop Singhal, and Latifur Khan. Alert: A framework for efficient extraction of attack techniques from cyber threat intelligence reports using active learning. In *IFIP Annual Conference on Data and Applications Security and Privacy*, pages 203–220. Springer, 2024.

[49] Priyanka Ranade, Aritran Piplai, Anupam Joshi, and Tim Finin. Cybert: Contextualized embeddings for the cybersecurity domain. In *2021 IEEE International Conference on Big Data (Big Data)*, pages 3334–3342. IEEE, 2021.

[50] Nanda Rani, Bikash Saha, Vikas Maurya, and Sandeep Kumar Shukla. Ttphunter: Automated extraction of actionable intelligence as ttps from narrative threat reports. In *Proceedings of the 2023 australasian computer science week*, pages 126–134. 2023.

[51] Nanda Rani, Bikash Saha, Vikas Maurya, and Sandeep Kumar Shukla. Ttpxhunter: Actionable threat intelligence extraction as ttps from finished cyber threat reports. *Digital Threats: Research and Practice*, 5(4):1–19, 2024.

[52] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

[53] Injy Sarhan and Marco Spruit. Open-cykg: An open cyber threat intelligence knowledge graph. *Knowledge-based systems*, 233:107524, 2021.

[54] Kiavash Satvat, Rigel Gjomemo, and VN Venkatakrishnan. Extractor: Extracting attack behavior from threat reports. In *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 598–615. IEEE, 2021.

[55] Yuval Schwartz, Lavi Ben-Shimol, Dudu Mimran, Yuval Elovici, and Asaf Shabtai. Llmcloudhunter: Harnessing llms for automated extraction of detection rules from cloud-based cti. In *Proceedings of the ACM on Web Conference 2025*, pages 1922–1941, 2025.

[56] Samaneh Shafee, Alysson Bessani, and Pedro M Ferreira. Evaluation of llm-based chatbots for osint-based cyber threat awareness. *Expert Systems with Applications*, 261:125509, 2025.

[57] SigmaHQ. github.com/SigmaHQ/sigma.

[58] Weihang Su, Yichen Tang, Qingyao Ai, Zhijing Wu, and Yiqun Liu. Dragin: Dynamic retrieval augmented generation based on the real-time information needs of

large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12991–13013, 2024.

[59] Weihang Su, Yichen Tang, Qingyao Ai, Junxi Yan, Changyue Wang, Hongning Wang, Ziyi Ye, Yujia Zhou, and Yiqun Liu. Parametric retrieval augmented generation. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1240–1250, 2025.

[60] Binhui Tang, Junfeng Wang, Huanran Qiu, Jian Yu, Zhongkun Yu, and Shijia Liu. Attack behavior extraction based on heterogeneous cyberthreat intelligence and graph convolutional networks. *Computers, Materials & Continua*, 74(1), 2023.

[61] R Vaitheeshwari, Eric Hsiao-Kuang Wu, Ying-Dar Lin, Ren-Hung Hwang, Po-Ching Lin, Yuan-Cheng Lai, and Asad Ali. Trace: Relationship analysis and causal factor extraction in cyber threat intelligence reports. *IEEE Transactions on Dependable and Secure Computing*, 2025.

[62] Xuren Wang, Xinpei Liu, Shengqin Ao, Ning Li, Zhengwei Jiang, Zongyi Xu, Zihan Xiong, Mengbo Xiong, and Xiaoqing Zhang. Dnrti: A large-scale dataset for named entity recognition in threat intelligence. In *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 1842–1848. IEEE, 2020.

[63] Zihan Wang, Ziqi Zhao, Yougang Lyu, Zhumin Chen, Maarten de Rijke, and Zhaochun Ren. A cooperative multi-agent framework for zero-shot named entity recognition. In *Proceedings of the ACM on Web Conference 2025*, pages 4183–4195, 2025.

[64] Ximing Wen, Wenjuan Tan, and Rosina Weber. Gaprotonet: A multi-head graph attention-based prototypical network for interpretable text classification. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 9891–9901, 2025.

[65] Ming Xu, Hongtai Wang, Jiahao Liu, Yun Lin, Chenyang Xu Yingshi Liu, Hoon Wei Lim, and Jin Song Dong. Intelex: A llm-driven attack-level threat intelligence extraction framework. *arXiv preprint arXiv:2412.10872*, 2024.

[66] Jingchen Yan, Zhe Du, Jifang Li, Shiduo Yang, Jinghao Li, and Jianbin Li. A threat intelligence analysis method based on feature weighting and bert-bigru for industrial internet of things. *Security and Communication Networks*, 2022(1):7729456, 2022.

[67] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[68] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, 32, 2019.

[69] Yara-Rules. Yara-rules. https://github.com/Yara-Rules/.

[70] Weiqiu You and Youngja Park. Cyber-attack technique classification using two-stage trained large language models. *arXiv preprint arXiv:2411.18755*, 2024.

[71] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. Bertscore: Evaluating text generation with bert. *arXiv preprint arXiv:1904.09675*, 2019.

[72] XiangRui Zhang, XueJie Du, HaoYu Chen, Yongzhong He, Wenjia Niu, and Qiang Li. Automatically generating rules of malicious software packages via large language model. In *2025 55th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 734–747. IEEE, 2025.

[73] Yongheng Zhang, Tingwen Du, Yunshan Ma, Xiang Wang, Yi Xie, Guozheng Yang, Yuliang Lu, and Ee-Chien Chang. Attackg+: Boosting attack graph construction with large language models. *Computers & Security*, 150:104220, 2025.

A Prompts & Examples

A.1 Information Extraction

(1) CommandX Construction Prompts. Templates used to generate diverse command-containing sentences for CommandX are provided online: **appendix/1**. (2) Command Extraction Prompts. Inference prompts for extracting commands on $\mathbb{D}_c$: **appendix/2**.

A.2 Attack Description Generation

(1) Example of Retrieved Knowledge. An example of external knowledge for technique T1562.001: **appendix/3**. (2) Knowledge Filtering Prompts. Prompts used to filter *CoA*, *CWE*, *CAPEC*, and *Mitigation*: **appendix/4**. (3) Description Generation Prompts. System prompt used in AttackDesc-DPO: **appendix/5**.

A.3 Sigma Rule Generation

A simple example of prompts for generating Sigma rules from commands and attack descriptions: **appendix/6**.