RulePilot: An LLM-Powered Agent for Security Rule Generation

Hongtai Wang[*]
National University of Singapore
Singapore, Singapore
wanghongtai0702@gmail.com

Ming Xu[*#]
National University of Singapore
Singapore, Singapore
mingxu@nus.edu.sg

Yanpei Guo
National University of Singapore
Singapore, Singapore
guo.yanpei@u.nus.edu

Weili Han
Fudan University
Shanghai, China
wlhan@fudan.edu.cn

Hoon Wei Lim
Cyber Special Ops-R&D, NCS Group
Singapore, Singapore
hoonwei.lim@ncs.com.sg

Jin Song Dong
National University of Singapore
Singapore, Singapore
dcsdjs@nus.edu.sg

Abstract

The real-time demand for system security leads to the detection rules becoming an integral part of the intrusion detection life-cycle. Rule-based detection often identifies malicious logs based on the predefined grammar logic, requiring experts with deep domain knowledge for rule generation. Therefore, automation of rule generation can result in significant time savings and ease the burden of rule-related tasks on security engineers. In this paper, we propose *RulePilot*, which mimics human expertise via LLM-based agent for addressing rule-related challenges like rule creation or conversion. Using *RulePilot*, the security analysts do not need to write down the rules following the grammar, instead, they can just provide the annotations such as the natural-language-based descriptions of a rule, our *RulePilot* can automatically generate the detection rules without more intervention. *RulePilot* is equipped with the intermediate representation (IR), which abstracts the complexity of config rules into structured, standardized formats, allowing LLMs to focus on generation rules in a more manageable and consistent way. We present a comprehensive evaluation of *RulePilot* in terms of textual similarity and execution success abilities, showcasing *RulePilot* can generate high-fidelity rules, outperforming the baseline models by up to 107.4% in textual similarity to ground truths and achieving better detection accuracy in real-world execution tests. We perform a case study from our industry collaborators, showcasing that *RulePilot* significantly help junior analysts/general users in the rule creation process.

CCS Concepts

• **Security and privacy → Software and application security**;

Keywords

LLM-based agents, Rule-based Intrusion Detection, Incident Response, AIOps

[*] Both authors contributed equally to the paper, ordered alphabetically.
[#] Corresponding author.

ICSE '26, Rio de Janeiro, Brazil

ACM ISBN 979-8-4007-2025-3/26/04
https://doi.org/10.1145/3744916.3773249

ACM Reference Format:
Hongtai Wang[*], Ming Xu[*#], Yanpei Guo, Weili Han, Hoon Wei Lim, and Jin Song Dong. 2026. *RulePilot*: An LLM-Powered Agent for Security Rule Generation. In *2026 IEEE/ACM 48th International Conference on Software Engineering (ICSE '26), April 12–18, 2026, Rio de Janeiro, Brazil.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3744916.3773249

1 Introduction

Security threats are increasingly a growing concern for both users and industrial organizations. The infamous SolarWinds attack [4] disrupted supply chains and compromised sensitive data, highlighting the critical need for robust security controls. A recent trend in intrusion detection systems relies on the neural-network-based provenance graphs, which have demonstrated notable strength in detection performance. However, they face the problems of high computational resource cost and long detection latency, hindering their wide practical deployment. In practice, in security detection systems, rules [11] are widely used to identify malicious activities and trigger alerts, such as detection rules executed on SIEM (Security Information and Event Management) platforms, which offer a lightweight and efficient solution to these challenges while maintaining great explanation abilities.

However, the high cost of rule creation and the long duration of rule maintenance are still problems faced by security organizations. Particularly, these detection rules are typically written manually by junior and senior security experts, a process that is time-consuming, labor-intensive and requires extensive domain knowledge. Furthermore, as attack techniques continue to evolve, rules need constant updates, increasing maintenance costs. Tools like MITRE ATT&CK [14] provide a common framework to describe attack techniques, but translating the structured techniques into specific rule configurations requires huge manual efforts. Moreover, modern security organizations can sometimes use different SIEM platforms such as Splunk [39], Microsoft Sentinel [27], or IBM QRadar [16], which have their own rule languages. Rules written for one platform cannot directly work on another, creating a cross-platform compatibility problem when an organization migrates an SIEM system. The automation of rule generation and conversion can result in significant time savings and ease the burden of rule-related tasks on security engineers.

The recent breakthrough of Large Language Models (LLMs), particularly in code generation [15, 21, 42, 47], text-to-SQL [37] and binary malware analysis [48, 52] with generative models like the GPT series, open new opportunities for automated security rule

generation and conversion. Unfortunately, compared to code/SQL generation, the challenge of generating security configurations lies in the nuanced and dynamic nature inherent in SIEM-specific rules. Code/SQL often follows well-defined syntax and logical structures, while rule configurations are highly domain-specific, system-dependent, and lack standardized formats. The rule configurations require a deep understanding of systems' behaviors and environment, precise tuning of parameters, dependencies, and iterative corrections, which can vary significantly across SIEM systems. Informally speaking, a junior programmer can write redundant but correct code, however, he/she may struggle to figure out a correct SIEM-specific rule constraint. Several works [35, 44] explored the generation of simple detection YAML rules like Sigma, falling short in addressing the more complex and functional challenges specific to SIEM systems due to the following challenges.

• **Non-standardized format:** The rule configurations are highly domain-specific and lack standardized representations. A standalone LLM typically lacks precise knowledge and cannot simulate human thoughts to break down the complex rule generation into smaller pipelines. To address this, we design an intermediate representation (IR) that can serve as a bridge between high-level requirements and low-level configuration file details. An IR abstracts the complexity of rule configurations into a structured, standardized format that captures essential parameters, and relationships, allowing LLMs to focus on generating configurations in a manageable and consistent way. The designed IR should be capable of handling the SIEM-specific cases like nested operators, vendor-specific syntax, reducing ambiguity and improving accuracy.
• **Iterative correction:** The initially generated rules might be semantically and syntactically incorrect, or logically-nonaligned. To resolve this, we introduce the reflection functions, identifying the potential mistakes upon each step, and refining the identified weakness to optimize the semantic and syntactic gaps. Beyond that, our reflection supports logical consistency, rule-field coverage, and the live execution viability, enabling the robust and scalable rule generation.
• **System dependence:** A sound rule should be able to interact with the live SIEM systems while existing LLMs fall short into autonomously and independently use tools like external SIEM vendor's grammar checks, feedback from live SIEM vendor's APIs, or rule-testing frameworks. We integrate the live Splunk [39] SIEM with LLMs, facilitating validation, optimization, and adaptation of configurations across systems and environments.

In this paper, for the first time, we propose *RulePilot*, which is an LLM-powered agent facilitating a series of practical scenarios on rule-based detection autonomously: 1) Using *RulePilot*, security analysts do not need to write rules following a specific grammar. Instead, they can simply provide annotations, such as rule descriptions in natural language. With this input, our *RulePilot* can automatically generate detection rules without requiring any further intervention. Usually, the descriptions can be divided into preconditions like a rule annotations or the attack types provided by experts. We tailor our workflow to Splunk SIEM grammars. 2) Furthermore, when security analysts update or migrate their SIEM systems, they need the conversion function between the different SIEM vendors. *RulePilot* supports the conversion between Splunk Processing Language (SPL) and Microsoft Sentinel Kusto Query Language (KQL).

We evaluate *RulePilot* upon objective similarity for textual alignment with ground truth rules [40] and semantic evaluator for assessing logical and functional correctness. Results show that *RulePilot* consistently improves both textual similarity and semantic accuracy, outperforming standalone LLMs by up to 107.4% in textual similarity. We conduct a field study by executing the generated rules in a realistic Splunk environment, evaluating their execution success in detecting suspicious activities. The results demonstrate that *RulePilot* successfully captures the majority of suspicious logs by up to 1.00 F1 score, validating its practical applicability in real-world threat detection scenarios. Our evaluation yields intriguing insights into the capabilities and limitations of LLMs in rule generation. We discover that LLMs show proficiency in understanding high-level threat descriptions and generating corresponding rules, however, we find that LLMs have difficulty in maintaining field mappings and condition handling, which necessitates human verification for checking the final results, ensuring the generated rule functions correctly. We perform a case study from our industry partners, and show that *RulePilot* significantly facilitates the rule generation of junior analysts/general users in terms of time used, rule quality including the syntax validity and logical alignment.

We summarize our contributions as follows.

- We propose a novel workflow *RulePilot* to address SIEM-specific rule generation and conversion challenges. Our designed immediate representation and reflection modular go beyond general IR and reflection mechanisms, effectively covering the SIEM-specific functions and edge cases such as nested operators and logical consistency, making the process more robust and scalable.
- We tailor our *RulePilot* to Splunk SIEM system, analyzing the grammars and environments specific to Splunk, seamlessly integrating with Splunk for intelligent and efficient rule execution. To the best of our knowledge, this is the first, end-to-end and real-time agentic framework for SIEM-rule creation.
- We conduct a comprehensive evaluation of *RulePilot*, employing models like GPT-4o, DeepSeek-V3, and LlaMa-3. *RulePilot* outperforms the baseline models by up to 107.4% in textual similarity to ground truths and achieves better detection accuracy in real-world Splunk execution tests.

With its structured reasoning and automation capabilities, *RulePilot* is poised to become an essential tool for security analysts in rule-relevant tasks. We release all the used datasets in the link [1] and open-source all code in [2].

2 Background and Motivation

2.1 Rule-based Anomaly Detection

Modern anomaly detection systems like Security Information and Event Management (SIEM) [11, 45] typically rely on detection rules to identify potential intrusions, which are widely used due to their lightweight overhead and great explanation abilities. The widely-used rules can be typically classified into the general Sigma and

[1] https://sites.google.com/view/rulepilot/dataset
[2] https://github.com/LLM4SOC-Topic/RulePilot

the SIEM-specific rules. Sigma is a generic and open signature format for SIEM systems, allowing for flexible rules in YAML format that can be translated into multiple SIEM vendors. Despite their compatibility with any SIEM vendor, Sigma rules primarily rely on single-pattern matching using regular expressions. They lack support for complex queries, such as SQL-style aggregations, and are unable to execute calculations including statistical analysis or time-window-based computations in SIEM vendors. These limitations often result in failures to detect sophisticated attacks involving a series of events or cycles. In contrast, SIEM-specific rules can bridge this gap by incorporating customized conditions with conditional statements and leveraging advanced functions. Consider a scenario where an attacker attempts to exfiltrate sensitive data by downloading multiple *".zip"* files from a server. A typical Sigma rule detects this behavior through pattern matching in log fields, such as identifying *".zip"* file requests in URI queries, which relies solely on string-based detection, lacking contextual awareness and deeper behavioral analysis. As a comparison, take the widely-used SIEM vendor Splunk [39] as an example, a Splunk-specific rule [40, 41] can implement the detection with more advanced functionality shown in Listing 1.

```
index=web_logs
| search uri_query="*.zip"
| stats count BY src_ip, uri_query, user_agent
| where count > 5 AND user_agent!="Mozilla/5.0 (friendly-
    bot)"
| eval message="Potential data exfiltration detected: " .
     src_ip . " downloading " . count . " ZIP files"
| table _time, src_ip, uri_query, user_agent, count,
    message
```

Listing 1: A splunk rule for detection of ZIP file downloads.

This Splunk rule can track activity over time, filter out events using the conditions (such as removing known bots), and generate meaningful alerts, making it far more effective in identifying attack behaviors. We commit to generating such SIEM-specific rules using our *RulePilot*. Among the SIEM vendors, we target Splunk vendors as Splunk [39, 40] is widely used by organizations in practice in literature and our professional experience. Additionally, we consider the problem of rule conversion when Splunk SIEM sometimes should be migrated into another SIEM like Microsoft Sentinel [27].

2.2 Motivation Scenarios

As shown in Figure 1, existing methods require analysts to write rules manually with an attack description. There are two types of analysts: senior analysts and junior analysts. A senior analyst has rich experience and years of writing rules. They can complete the task in a short time, and the rules are of good quality. However, the cost is very high due to training and salaries. A junior analyst may lack experience. They take a long time to write rules, and the results may not be good. This motivates the use of a *RulePilot* based on LLMs. *RulePilot* assists in writing rules, saving time, reducing costs, and achieving better results, only with the help of junior analysts for somewhat condition handling. Companies often encounter system migration challenges, such as adapting validated rules to a new SIEM platform. *RulePilot* addresses this by offering a rule conversion function, enabling seamless rule adaptation across different SIEM systems (e.g., from Splunk to Microsoft Sentinel). This ensures that rules generated by *RulePilot* remain reusable, minimizing manual effort and streamlining future migrations. Note that *RulePilot* is designed to generate rules autonomously without human intervention. We acknowledge that human verification remains essential for final deployment. In practice, human operators validate the generated rules to ensure their correctness and effectiveness. The junior operator here is expected to be familiar with the SIEM environments. Compared to manual rule creation, the human role here focuses on validation, eliminating the need to master complex rule grammars.

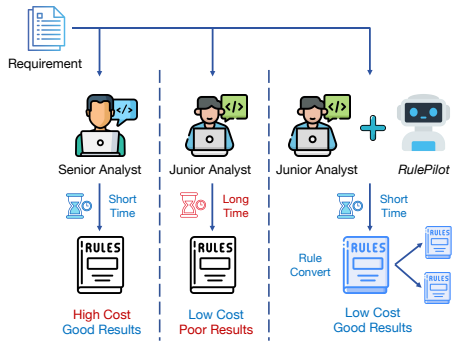


Figure 1: Motivation scenario [6, 18]: By combining with *RulePilot*, junior analysts can generate concise detection rules and conversions, significantly reducing the workload of senior experts.

2.3 LLM-Based Agents

An agent can be broadly defined as an autonomous entity that perceives its environment, makes decisions based on its goals, and takes actions to affect its surroundings[33]. These features are inspired by human cognition and allow agents to behave consistently and effectively in dynamic, complex environments [46]. Researchers have been exploring machine learning techniques to automate aspects of rule generation in cybersecurity. For example, Raff et al. [30] introduced AutoYara, a tool that utilizes biclustering algorithms to automatically generate YARA rules for malware detection. In another study, Saxe [34] developed YaraML, a machine learning-based toolkit designed to automate the creation of YARA rules. However, they always focus on generic YARA rules, while ignoring the customized SIEM-specific rules. Applying LLM-based agents to generate complex detection rules, such as Splunk-specific rules, faces several significant challenges: 1) deep domain knowledge requirement. This involves meticulously analyzing rule structures step by step and crafting modular designs to guide the LLM in making precise plans and reasoned decisions. and 2) workflow design and live SIEM integration. Developing an agent workflow capable of handling multi-step reasoning and integrating the SIEM systems is equally demanding.

3 *RulePilot*: Methodology

3.1 Workflow Design

Overview. As shown in Figure 2, *RulePilot* consists of three key components: Chain of Thought (CoT) reasoning, Intermediate Representation (IR), and Reflection & Iterative Optimization. Given an

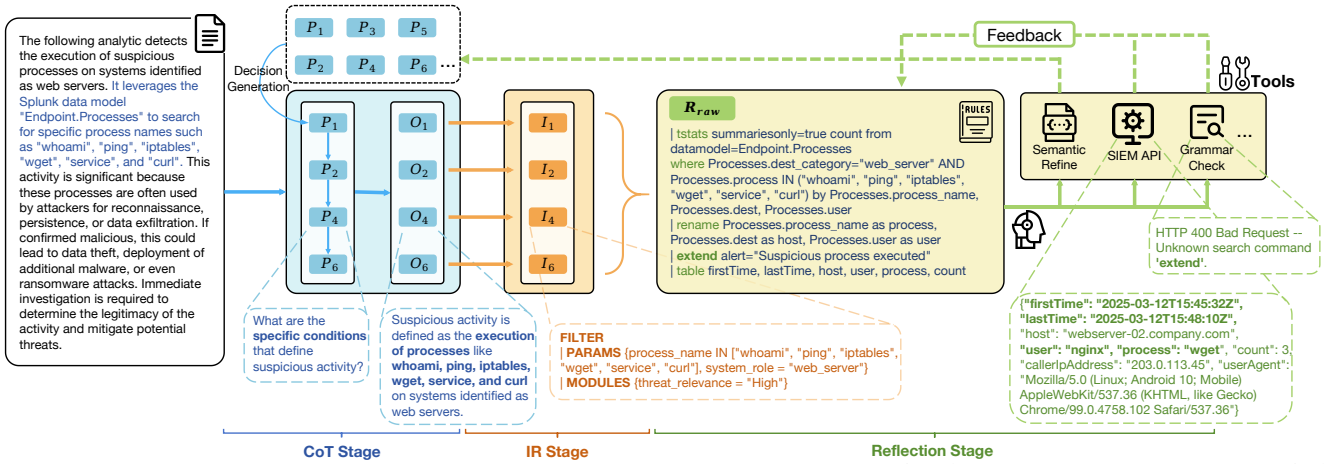


Figure 2: Overview of *RulePilot*, where *RulePilot* incorporates the Chain-of-Thought (CoT), Intermediate Representation (IR), and Reflection components. Given the NLP-based input requirements, *RulePilot* generates SIEM-specific rules that can be directly executed in the SIEM to detect malicious logs and trigger alerts (i.e., the malicious log).

initial rule generation request R, *RulePilot* applies Least-to-Most Prompting (LMP) [53] to incrementally decompose rule generation into a structured sequence of reasoning steps P. Each step produces an intermediate representation that encodes its core logic, and once all steps are processed, these representations are aggregated to form an initial rule R_{raw}. The rule then undergoes reflection and iterative optimization, where weaknesses are identified and refined through reasoning adjustments, syntax validation, and execution-based feedback. This structured analyze-generate-reflect-optimize cycle ensures that the final rule R_{final} is both logically sound and execution-efficient.

Chain of Thought Reasoning. The predefined CoT reasoning steps that *RulePilot* can select and execute are below:
① *Interpreting the security objectives,* ② *Identifying data/log sources,* ③ *Defining initial filters/conditions,* ④ *Extracting relevant fields,* ⑤ *Performing data aggregation,* ⑥ *Optimizing the rule*

Given a rule generation request R, *RulePilot* first decomposes it into a sequence of structured reasoning steps $P = \{p_1, p_2, ..., p_n\}$. The decomposition follows these two stages: (1) Step Selection: The model selects relevant steps from the set of predefined steps above, which cover the comprehensive aspects of rule generation. (2) Stepwise Execution: The model answers each subproblem in sequence, using the output of p_i as contextual input for p_{i+1}, ensuring a gradual and structured rule refinement process.

Intermediate Representation. A well-designed IR abstraction provides a clear and structured way to express rule intention. SIEM-specific rule logic is inherently complex, requiring structured filtering, aggregation, and anomaly detection across various log sources. The syntax of SIEM rule languages, such as Splunk Processing Language (SPL) and Microsoft Sentinel Kusto Query Language (KQL), is highly complex, making direct generation challenging. By incorporating an intermediate representation, *RulePilot* enables the model to prioritize semantic logic over syntactic details, streamlining the reasoning process and improving rule generation accuracy.

Specifically, each reasoning step corresponds to one or more IR statements, with each IR statement representing a single processing unit (pipe) in the rule. Given an input rule request description, the CoT process generates a sequence of reasoning steps $P = \{p_1, p_2, ..., p_n\}$, where each step p_i maps to an IR component I_i. Formally, this can be represented as:

$$I = \{I_1, I_2, ..., I_n\} = \bigcup_{i=1}^{n} \mathcal{T}(o_{p_i})$$

where $\mathcal{T}$ is the transformation function that maps each intermediate analysis output o_{p_i} to an IR I_i. The final IR set I is the union of all individual IR components generated through this mapping process.

Once the full set of IR components I has been constructed, the initial raw rule R_{raw} is generated by integrating both the semantic insights from o_{p_i} and the structured transformations from I_i.

$$R_{raw} = C(O, I)$$

where C is the rule construction function that synthesizes the intermediate analyses O and the IR I into an executable rule.

Reflection and Iterative Optimization. The optimization process incorporates a dynamic iterative mechanism, starting with an automated reflection function $\Phi(R_{raw})$, which analyzes the generated rule to identify logical inconsistencies, knowledge gaps, or syntax errors. After each iteration, the *RulePilot* autonomously selects and invokes appropriate tools to address deficiencies. If the system detects unresolved issues, it triggers another refinement cycle, iterating until the rule is logically coherent, semantically accurate, and syntactically valid.

$$S = \{s_{lc}, s_{sc}, s_{ev}\} = \Phi(R_{raw})$$

where S represents the set of identified issues:

- Logical Consistency (s_{lc}): Verifies whether all filtering, aggregation, and correlation steps align with the intended detection logic.
- Syntax Correctness (s_{sc}): Confirms that the rule adheres to the syntax requirements of the target SIEM system, such as Splunk SPL or Microsoft KQL.
- Execution Validity (s_{ev}): Ensures that the rule is structured for efficient query execution without excessive computational overhead.

3.2 Detailed Construction

Here, we show how to tailor our workflow to the Splunk SIEM, generating SPL rules.

Chain of Thought Reasoning. We keep the core CoT workflow unchanged, and incorporate additional Splunk-adapted design elements to improve log source selection, syntax correctness, and execution efficiency. A detailed example of the CoT prompt structure is shown in Table 1. We open-source the prompts for LLMs corresponding to each function of these processes on the website [32]. Our prompt strategy is motivated by empirical tuning and expert insights, effectively mimicking expert-level expertise into the agent's behavior. We used components of Identity, Instructions, Examples, and Context because they reflect how SIEM experts retrieve historical cases during manual rule construction. We also guide with DOs and DON'Ts, helping the model understand both what to do and what to avoid, based on OpenAI's official guide [3].

Table 1: Structure of the Prompt Template

CoT Prompt Template
You are a security analyst at a cybersecurity company, specializing in writing and optimizing Splunk rules (SPL) for threat detection. Task: <Iterate through the tasks in the Task List> Instruction: <Specific Guidance such as possible keywords> Example Input: <Provide an example rule description> Example Output: <Corresponding SPL detection rule>
Task List
① Map security objectives to Splunk event sources ② Determine necessary log fields ③ Define efficient filtering conditions ④ Apply field extractions and transformations ⑤ Perform aggregations and anomaly detection ⑥ Optimize query execution and validate syntax

Intermediate Representation. We create the IR structure tailored to Splunk SPL, whose structure follows a three-part format below.

$$<KEYWORD>|PARAMS\{k_i = v_i\}|MODULES\{m_j\}$$

where:

• *KEYWORD* represents the core function of each rule step, including filtering logs, extracting fields, performing aggregations, or applying transformations. These IR keywords are summarized based on an extensive analysis of open-source Splunk rule sets and proprietary rules from industry collaborators. Each *KEYWORD* corresponds to one or more SPL commands. Table 3 presents the distribution and frequency of the 15 predefined IR keywords in SPL, along with their associated SPL commands.

• *PARAMS* serves as the core configuration of the rule, defining mandatory elements such as log sources, filtering conditions, and time constraints. These parameters ensure that the rule is executed within the correct context. For example, specifying *index="auth_logs" source="WinEventLog:Security"* ensures that the rule retrieves logs from relevant data sources, preventing inefficiencies caused by querying unrelated logs. Similarly, including a time constraint like *earliest=-15m latest=now* helps narrow the search scope, significantly improving query speed.

• *MODULES* introduces functional annotations that enrich the interpretation of a rule, improving its flexibility, readability, and adaptability during the transformation into an executable query. Unlike *PARAMS*, which strictly define the necessary elements for rule execution, *MODULES* describe the intended logic and analytical operations that should be applied to the retrieved data. For example, a module may specify "track user behavior across sessions" or "identify repeated failed login attempts", helping to capture the intent behind the rule rather than just its execution details.

Table 2 shows an example on how our IR corresponds to an executable Splunk query, where the IR abstraction defines the detection logic in a structured and interpretable way, with its SPL counterpart represents the actual execution in Splunk. The *FILTER* specifies where to retrieve logs, *PARAMS* ensures correct data scoping and *MODULES* encapsulates the detection intent, guiding how the rule should process events.

Table 2: Example for IR Statement to a Splunk Rule Pipe

IR Example
FILTER | PARAMS {index="auth_logs", source="WinEventLog:Security", earliest=-30m} | MODULES {"Aggregate login attempts", "Detect brute force login attempts"}
Corresponding Splunk SPL Pipe
index="auth_logs" source="WinEventLog:Security" earliest=-30m | stats count by src_ip | where count >10

To address SIEM-specific edge cases, we design the IR with explicit support. For nested operators, which are commonly used in filter and transformation logic, we incorporate two strategies: For simple constructs (e.g., eval, match, where), we allow controlled

Table 3: IR keywords along with their SPL Commands.

Keyword	SPL Command	Frequency
FILTER	search, where, eval, match	3129
EXTRACT	rex, spath, extract, kv	2063
AGGREGATE	stats, timechart, eventstats, tstats	1872
OUTPUT	table, fields,outputlookup, return	1609
TRANSFORM	eval, replace, convert, fillnull	1015
RENAME	rename	802
LOOKUP	lookup,inputlookup,outputlookup	315
BUCKET	bin, bucket	91
JOIN	join, appendcols, transaction	83
FILL	fillnull, coalesce, replace	75
APPEND	append, union, appendpipe	46
SORT	sort, reverse	38
DEDUP	dedup, uniq	28
APPLY	apply, fit	24
DEBUG	noop, logtrace, dump, sendemail	17

[3] https://platform.openai.com/docs/guides/text?api-mode=chat

nesting within a single IR statement; for more complex expressions involving multi-layer joins or condition chaining, we enforce decomposition into multiple atomic IR statements to preserve interpretability and reduce error propagation during transformation.

To accommodate SIEM-specific syntax variations of multiple SIEMs, our IR incorporates a pluggable keyword dictionary architecture. For each target SIEM system (e.g., Splunk SPL, Microsoft KQL), we can curate and maintain a dedicated dictionary of IR keywords and associated translation templates derived from empirical rule corpora and vendor documentation. This allows the IR-to-query compiler to flexibly adapt the output semantics and syntactic form of different SIEMs.

Reflection and Iterative Optimization. Different from LLM-based self-debugging [13, 43] that typically relies on prompting-based re-generation conditioned on observed errors or exceptions, our reflection adopts a scoring-based multi-layered mechanism. This enables the system to iteratively reconstruct faulty rule components at both the IR and final SPL levels, addressing deeper issues such as semantic gaps, abstraction mismatches, logical inconsistencies, field coverage, and execution viability, rather than merely rewriting surface text. Our reflection mechanism integrates semantic-level diagnostics, real execution feedback via selective modules, and a scoring-based evaluation to further analyze the logical consistency, field coverage, and execution viability using a scoring-based evaluation (s_{lc}, s_{sc}, s_{ev}), and selectively invokes CoT-based refinement and SIEM-integrated validation routines.

Specifically, *RulePilot* first performs syntax validation using Splunklib's dry-run mode [38], which allows the system to check for syntax errors without executing the query. Second, *RulePilot* dynamically invokes a set of predefined optimization modules to address identified logical/structural inconsistencies. If $\Phi(R_{raw})$ detects any of $\{s_{lc}, s_{sc}, s_{ev}\}$, indicating logical inconsistencies, structural violations, or execution failures, the system applies targeted refinements through two steps:

(1) CoT-Based Refinement Modules (M_{CoT}): The system revisits earlier CoT reasoning steps and regenerates the affected IR statements, refining the rule logic and improving structural coherence. This produces an updated intermediate rule R'_{raw}, defined as:

$$R'_{raw} = M_{CoT}(R_{raw}, s_{lc})$$

(2) Rule Refinement Modules (M_{Splunk}): *RulePilot* integrates with live Splunk's validation and execution environment. The rule is executed via Splunk's API, retrieving real log data. If the results deviate from the intended detection objective, *RulePilot* iteratively refines the SPL by adjusting filters, modifying conditions, or restructuring query logic based on execution feedback.

$$R_{final} = M_{Splunk}(R'_{raw}, S)$$

3.3 Rule Conversion

To ensure the interoperability and adaptability of *RulePilot* across different SIEM platforms, we implement a Rule Conversion Module that translates rules from one SIEM vendor (e.g., Splunk SPL) into another (e.g., Microsoft KQL). This conversion process is designed to preserve the logical intent of the original rule while adapting it to the syntax, function names, and query structures of the target SIEM.

As shown in Algorithm 1, the Rule Conversion follows a structured multi-step approach, beginning by segmenting the input rule into individual pipes, and then breaking down a complex task into smaller, more manageable units. However, this breakdown may cause a loss of contextual dependencies between pipes, so an LLM-based function extraction module is introduced to retrieve each pipe's purpose and variable mappings, ensuring coherence in later stages. Once the semantic information is extracted, the system converts each pipe sequentially, allowing previously processed pipes to provide contextual support.

To further improve accuracy, we incorporate a Retrieval-Augmented Generation (RAG) [19] mechanism that dynamically maps keywords and functions from the source SIEM to their equivalents in the target SIEM. The RAG knowledge base is bootstrapped from Microsoft's official migration documentation [7], which provides detailed mappings between Splunk detection rules and their KQL counterparts. From this corpus, we extract every `<SPL command, KQL operator, usage snippet>` triple, normalise aliases (e.g., `rex` ↔ `extract`) and build a key-value mapping database that aligns functionally equivalent operations across the two SIEMs. Each SPL command string and its descriptive context are embedded with the `text-embedding-ada-002` [3]. During conversion, every pipe is first tokenised into *<verb, args, fields>* tuples. The verb plus surrounding comments are embedded on-the-fly, and a top-k (default $k = 10$) vector search is issued. Candidates with cosine similarity above a threshold (i.e., 0.82 used) are retained and re-ranked with a BM25 [31] lexical score to favour exact-string matches. If the SPL command has a high-confidence match, the retrieved KQL operator (and an example usage) is attached to the LLM prompt as a structured "conversion hint". This retrieval-augmented approach ensures that the LLM does not solely rely on pre-trained knowledge but is instead guided by vendor-specific best practices and real-world rule patterns.

Algorithm 1 Rule Conversion from SIEM Vendor A to B

Require: Rule R_A from SIEM Vendor A, Target SIEM Vendor B
Ensure: Converted rule R_B for SIEM Vendor B
Step 1: Pipe Segmentation
Split R_A into a sequence of pipes: $P = \{p_1, p_2, ..., p_n\}$
Step 2: Function Extraction
for each pipe $p_i \in P$ **do**
  $(f_i, in_i, out_i) \leftarrow \text{ExtractFunctionInfo}(p_i)$
end for
Step 3: Context-Aware Pipe Conversion
for each pipe $p_i \in P$ **do**
  $P_{\text{prior}} \leftarrow \text{GetPriorPipes}(p_i, P)$
  $K_i \leftarrow \text{RetrieveKeyword}(p_i, \text{Vendor A, Vendor B})$
  $p'_i \leftarrow \text{ConvertPipe}(p_i, in_i, out_i, P_{\text{prior}}, K_i)$
  $P' \leftarrow P' \cup \{p'_i\}$
end for
Step 4: Assemble Converted Rule
$R_B \leftarrow \text{AssembleRule}(P')$
return R_B

4 Evaluation

In this section, we aim to evaluate the following research questions.

- RQ1-**Accuracy**: How effective is *RulePilot* in generating SIEM-specific detection rules, measured by similarity to official rules and execution success across SIEM vendors?

• RQ2-**Efficiency**: What are the latency and resource costs of the rule generation process?
• RQ3-**Ablation Study**: Do the specific components in *RulePilot* help improve the quality of rule generation?
• RQ4-**Compatibility**: Does *RulePilot* support the conversion between Splunk SPL and Microsoft KQL?

4.1 Experimental Settings

4.1.1 Implementation Details. We implemented a fully functional prototype of *RulePilot*, designed for automated rule generation and conversion in Splunk SIEM. *RulePilot* is built upon GPT-4o, DeepSeek-V3 (671B), and LLaMA-3 (405B), with agent-based orchestration implemented using their function-calling capabilities. To control generation behavior, we configure all models with a temperature of 0.3 (balancing determinism and flexibility), top-p of 0.9 (ensuring controlled diversity), and set a maximum response length of 512 tokens to prevent excessively long outputs. We use the Splunk of version 9.3.1, and the trial license for experiments.

4.1.2 Datasets. We evaluate *RulePilot* using two sources of rules: **Splunk official rules** from the Splunk Security Content repository [4] as the ground truth to evaluate the similarity score, and the **custom rules between Splunk SPL and Microsoft KQL currently used by our industry collaborator** for their security operations to evaluate the compatibility. The Splunk official rules we download contain a total of 1,699 samples, organized into five major categories based on their focus (shown in Table 4), maximizing coverage across comprehensive and diverse categories, reducing evaluation bias. The key components of the datasets include the rule body (SPL) and the corresponding description that explains the purpose and context of the rule. The original datasets contained macros, which are vendor-specific or environment-specific variations. The macro abstracts the specific directives, and do not conform to rule grammars, possibly affecting rule consistency. To ensure a fair comparison, we replaced all macros with standardized definitions based on Splunk's official macro library. For example, a macro like `'process_cmd'`, should be replaced with its specific SPL equivalent: `Processes.process_name = cmd.exe`.

4.1.3 System Log Collection. To evaluate the execution success of our generated rules, we simulate various atomic attacks [1] provided by MITRE ATT&CK, including 229,968 system logs from 61 atomic tests, covering 12 tactics in MITRE ATT&CK, covering broad applicability and reducing evaluation bias. The system logs were collected using EventViewer in a controlled environment, capturing system, Sysmon, and PowerShell logs on a virtual machine running Windows 10 (64-bit). Our evaluation focuses primarily on Windows events due to their widespread use in both enterprises and consumer marketss [12]. The datasets have their labels based on our simulation process, open-sourced in [32].

4.1.4 Baseline. We compare *RulePilot*'s performance against that based upon the standalone LLMs of GPT-4o, DeepSeek-V3 (671B), and LLaMa-3 (405B), without the structured reasoning and function-calling mechanisms of *RulePilot*. For a fair comparison, both baseline models generate rules using the same prompts as those employed by *RulePilot* in its rule generation step, without any additional multi-step processing, validation, or refinement. We download DeepSeek-V3 and LLaMa-3 models from Hugging Face [5],[2] To ensure consistency across all models, we use the same model parameters as those employed by *RulePilot*.

[4]https://github.com/splunk/security_content. We primarily use rules from the *detections* folder to evaluate the similarity score and reference macro definitions from the *macros* folder for completeness.

Table 4: Category of our ground-truths, with each item containing the NLP descriptions and associate SPL rules.

Rules-Set Type	size	Time Frame
Application	125	2024-09-30 – 2024-11-19
Cloud	271	2024-09-30 – 2024-10-31
Endpoint	1187	2024-09-24 – 2024-12-03
Network	44	2024-09-25 – 2024-11-06
Web	72	2024-09-30 – 2024-10-17

4.2 Evaluation Metrics

Accuracy. Our accuracy evaluation consists of two complementary metrics: *quantifiable similarity assessment* and *an LLM-based evaluator*. The first metrics measure the textual similarity between the generated rules and the ground truth (i.e., the official rules), creating an objective assessment of structural and lexical alignment. We adopt three well-established quantifiable metrics below.

• *ROUGE* (Recall-Oriented Understudy for Gisting Evaluation) [22] is an NLP metric that compares machine-generated text with reference text to measure content similarity. A higher ROUGE-k indicates a greater overlap of k-grams between the generated and ground truth rule. Here, we set k = 1 and include ROUGE-L, capturing the longest common subsequence to reflect structural alignment.
• *BLEU* (Bilingual Evaluation Understudy) [28] is a widely used precision-oriented NLP metric that evaluates text similarity based on n-gram overlap. A higher BLEU-k score indicates better alignment between the generated and ground truth rule. We use BLEU-4 in our evaluation.
• *METEOR* (Metric for Evaluation of Translation with Explicit ORdering) [9] is an advanced NLP metric that improves upon BLEU by incorporating stemming, synonym matching, and word order considerations, making it a more robust metric for comparing variations in rule expressions.

The *quantifiable similarity assessment* offers an objective metric but may yield misleadingly high scores for syntactically similar yet semantically incorrect rules. To mitigate this, we adopt the LLM-as-a-judge approach, a scalable and explainable method for approximating human preferences [20]. We evaluate rule quality from a semantic perspective, considering six key evaluation dimensions below.

• *Logical Consistency* (LC). The LLM looks at conditions, operators, and filters to see if anything is missing or changed.
• *Syntax Correctness* (SC). The LLM checks for mistakes in the query and looks for ways to write it better.
• *Readability & Maintainability* (RM). The LLM checks if the rule is written in a clear way, without unnecessary complexity.
• *Condition Coverage* (CC). The LLM ensures no key conditions are missing or unnecessary constraints are added.

• *False Positive & False Negative Risk* (FPFNR). The LLM checks if the rule is too strict (which may miss real threats) or too loose (which may flag normal activities).
• *Execution Efficiency* (EE). The LLM analyzes whether the rule uses complex operations, unnecessary filters, or inefficient queries that could slow down processing.

We adopt a scoring scheme ranging from 0 to 1 for each evaluation dimension. Instead of focusing on absolute scores, we emphasize relative rankings across outputs under the same prompt for evaluating the effectiveness of different methods. To mitigate evaluation bias, we followed a human-aligned iterative evaluation framework. An experienced human expert and an LLM were involved in an iterative prompt refinement process to align the evaluation standards. We define inter-rater agreement as a match in relative preference, for example, both the human and the LLM giving higher scores to *RulePilot* over the corresponding vanilla LLMs (baseline) is considered consistent, regardless of exact numerical values. Under this definition, we show the matrix in Figure 3 based on pairwise preferences, where our inter-rater agreement test reaches a larger Cohen's Kappa [17] score of 0.85, indicating strong agreement.

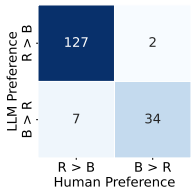


Figure 3: Preference inter-rater alignment between the LLM and the human evaluator. R refers to *RulePilot*, and B refers to Baseline. The ">" indicates that the method was rated higher in a pairwise comparison. The diagonal entries indicate agreement between the LLM and the human evaluator.

Execution Success. We ingest all collected logs into Splunk, execute the generated rules as search queries, and verify their accuracy by comparing retrieved logs against the ground truth from simulated atomic tests. Execution is considered successful if the retrieved logs match the expected attack-generated logs. We compute precision, recall, and other metrics to assess rule effectiveness. To quantify performance, we use $precision = \frac{TP}{TP+FP}$, $recall = \frac{TP}{TP+FN}$, and $F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision}+\text{Recall}}$ as evaluation metrics.

4.3 Evaluation Results

4.3.1 RQ1-Accuracy. We present our *quantifiable similarity assessment* in Table 5. Overall, *RulePilot* consistently outperforms all its corresponding baseline models across every category. The improvements range from 20.9% to 107.4%, demonstrating that *RulePilot* significantly enhances both the syntactic accuracy of the generated rules. Among the different detection categories, the cloud achieves the highest overall performance, suggests that cloud-based detection rules are relatively easier for *RulePilot* to generate accurately, possibly due to the structured and well-documented nature of cloud security rules. In contrast, the web category exhibits the lowest performance across most metrics. This indicates that web-related security rules tend to be more complex or diverse, making it harder for models to capture accurate patterns. Literature [8], [10] also support this claim that web-related security rules are complex. When comparing different LLMs, *RulePilot* achieves its highest performance using GPT-4o, compared to and LLaMa-3. This suggests that GPT-4o is better suited for structured rule-generation tasks, likely due to its improved reasoning and instruction-following capabilities.

Baseline Model Output

```
index=aws_cloudtrail sourcetype=\"aws:cloudtrail\"
(eventName=\"CreateLoginProfile\" OR eventName=\"ConsoleLogin\")
| eval eventType=case(eventName==\"CreateLoginProfile\", "Profile
Creation\", eventName==\"ConsoleLogin\", "Login Attempt\")
| stats count by userName, srcIp
| where count > 1
| sort _time
| table _time, userName, srcIp, count
```

***RulePilot* Output**

```
index=aws_cloudtrail sourcetype=\"aws:cloudtrail\"
(eventName=\"CreateLoginProfile\" OR eventName=\"ConsoleLogin\")
| stats count by sourceIPAddress, eventName,
requestParameters.userName, _time
| where eventName=\"CreateLoginProfile\" AND
eventName=\"ConsoleLogin\"
| transaction sourceIPAddress startswith=CreateLoginProfile
endswith=ConsoleLogin maxspan=5m
| table _time, eventName, userAgent, errorCode,
requestParameters.userName
```

Figure 4: Expert-reviewed comparison between rules generated by *RulePilot* versus those generated by standalone GPT-4o.

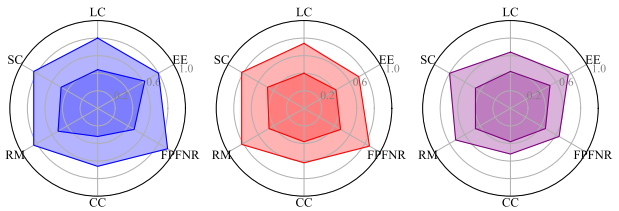
Looking deeply, our experts judge a concrete example of a rule generated by *RulePilot* and the GPT-4o baseline model. We present an expert-reviewed result in Figure 4. This demonstrates a rule related to AWS CloudTrail login and profile creation monitoring. We can find that the baseline model fails to properly correlate profile creation and subsequent login attempts, relying only on counting occurrences per user and IP. While it correctly filters relevant event types, it lacks a mechanism to determine if a login actually follows a profile creation within a short window.

Second, to avoid the syntactically similar yet semantically incorrect evaluation, we show the results of radar chart in Figure 5, illustrating the results of LLM-based evaluator results. We can find that *RulePilot* consistently outperforms all standlone LLMs across six evaluation dimensions. GPT-4o achieves the highest performance across most dimensions, particularly in Syntax Correctness (SC). In contrast, DeepSeek-V3 and LlaMa-3 show weaker performance, especially in Condition Coverage (CC) and False Positive & False Negative Risk (FPFNR).

Execution Success. To show the execution success of the generated rules in a SIEM vendor, we show our results in Table 6, where we find that *RulePilot* consistently achieves higher precision and recall across most tactics compared to GPT-4o, demonstrating its ability to generate executable detection rules in Splunk. Notably, *RulePilot* achieves perfect (100%) precision across multiple tactics, confirming that its generated rules accurately match ground truth detections without false positives in these cases. However, certain tactics exhibit lower precision and recall, particularly for Privilege Escalation, where GPT-4o fails entirely (0% precision and recall), while *RulePilot* retains some detection capability. These lower scores

Table 5: Syntax-level evaluation. Similarity comparison of the generated rules from *RulePilot* (RP) and baselines (BL) with ground truth, including GPT-4o (GPT), DeepSeek-V3-671B (DS), and LLaMa-3-405B (LLaMa).

Category	BLEU (↑)						ROUGE-1 (↑)						ROUGE-L (↑)						METEOR (↑)					
	GPT		DS		LLaMa		GPT		DS		LLaMa		GPT		DS		LLaMa		GPT		DS		LLaMa	
	RP	BL	RP	BL	RP	BL	RP	BL	RP	BL	RP	BL	RP	BL	RP	BL	RP	BL	RP	BL	RP	BL	RP	BL
application (54)	39.1	33.6	43.8	30.2	32.5	31.1	49.2	36.6	42.3	33.9	41.6	36.8	41.7	26.2	32.3	24.4	33.2	26.5	41.3	27.3	26.7	19.6	29.5	27.1
cloud (271)	47.9	33.9	40.9	25.4	38.3	27.0	58.7	44.4	53.1	21.6	48.1	28.8	53.7	37.5	52.5	29.1	46.1	26.7	58.1	43.8	61.4	32.2	65.0	33.9
endpoint (1,187)	42.8	29.5	34.8	24.5	32.8	27.8	59.8	36.6	51.0	24.4	48.7	43.0	57.3	32.5	42.0	30.9	40.1	37.0	66.3	37.8	35.9	29.6	42.5	22.9
network (43)	41.9	41.5	35.2	27.9	45.4	39.0	60.1	49.5	37.0	27.5	58.8	37.7	57.1	43.3	27.0	18.4	58.2	30.0	59.2	42.4	25.5	24.9	60.7	34.9
web (72)	41.6	34.8	27.5	22.1	32.0	28.2	57.0	43.3	39.8	20.6	38.6	28.8	50.1	37.9	31.9	14.5	36.6	27.3	56.6	41.2	28.9	17.4	43.8	34.8
Total (1,627)	43.4	30.9	35.8	24.8	34.0	28.1	59.1	38.5	50.2	24.2	48.2	39.7	55.9	33.6	42.6	29.3	41.2	34.3	63.5	38.7	39.3	29.0	46.4	25.7



(a) GPT-4o (b) DeepSeek-V3 (c) LlaMa-3

Figure 5: Semantic-level evaluation. Radar chart of LLM-based evaluator: the inner shaded area represents the baseline model's score, while the outer contour represents *RulePilot*'s score.

are primarily due to the *subtle* nature of malicious logs associated with these tactics, where critical identifying fields do not explicitly appear in the rule descriptions. As a result, baseline models struggle to generate effective detection rules, relying only on static descriptions without real-time feedback.

In contrast, *RulePilot* demonstrates significantly better performance due to its ability to autonomously call the Splunk API, retrieving real-time log feedback and refining its rules iteratively. This self-reflective and API-driven approach enables *RulePilot* to detect complex attack patterns that GPT-4o fails to capture, particularly in scenarios where key indicators are not directly stated in the initial rule descriptions.

Failure Cases. We analyze that the failures often occur when the key behavioral indicators are implicitly described in input descriptions, rendering *RulePilot* and vanilla LLMs ineffective. For example, we have checked the low recalls of 0.21 (*RulePilot*) and 0.0 (GPT-4o) in our own tests of Privilege-Escalation rules, the input description states "These calls are used to spawn MSBuild.exe in a suspended state before injecting the decrypted SaintBot binary into it, modifying the thread context to point to the malicious entry point and resuming the process" without the behavioral indicators of process hollowing- a technique often used for privilege escalation or execution evasion. Under the same inputs, *RulePilot* can consistently outperform the vanilla GPT-4o.

Answer to RQ1: *RulePilot* agent-based reasoning mechanism enhances logical structure and syntax adherence in terms of both similarity score and the execution success.

Table 6: Execution-level success on the Splunk SIEM.

Tactic	Precision (%)		Recall (%)	
	RulePilot	GPT-4o	*RulePilot*	GPT-4o
Reconnaissance	1.000	1.000	1.000	1.000
Initial Access	1.000	1.000	1.000	1.000
Execution	1.000	0.909	0.750	0.416
Persistence	1.000	1.000	0.818	0.714
Privilege Escalation	0.600	0.000	0.214	0.000
Defense Evasion	0.733	0.600	0.833	0.656
Credential Access	1.000	1.000	0.450	0.264
Discovery	0.667	0.167	0.444	0.100
Lateral Movement	1.000	1.000	0.667	0.667
Collection	1.000	1.000	1.000	1.000
Command and Control	1.000	1.000	1.000	1.000
Exfiltration	0.667	0.333	0.500	0.200
Impact	0.722	0.594	0.650	0.731

Table 7: Efficiency and cost of *RulePilot* and baselines.

Model		Prompt Tokens	Output Tokens	Money Cost	Generation Time
GPT-4o	*RulePilot*	13,752	2489	$0.060	78s
	Baseline	1,295	325	$0.012	12s
DeepSeek-V3	*RulePilot*	24,820	4,296	–	158s
	Baseline	3,284	772	–	31s
LLaMA-3	*RulePilot*	22,107	2,985	–	119s
	Baseline	1,734	474	–	26s

4.3.2 RQ2-Efficiency. We present the computational and economic costs in Table 7, which are derived by running *RulePilot* and the baseline approach on Splunk's open-source datasets (detailed in Table 4) and averaging the results across multiple test cases. We find that *RulePilot* requires more tokens and computation time than the baseline approach, mainly due to its stepwise reasoning and iterative refinement. However, this also results in more complete and logically structured rules, as seen in earlier evaluations, with accessable latency.

Answer to RQ2: *RulePilot* generates well-structured rules while maintaining accessible latency.

4.3.3 RQ3-Ablation Study. To further evaluate the effectiveness of the key components in *RulePilot*, we conduct an ablation study focusing on two critical elements: the IR and the combination of CoT reasoning and Reflection (CoT-R). Since Reflection involves iterative refinements that call CoT modules, these two components are inherently linked and evaluated as CoT-R. To assess the individual contributions, we introduce three experimental variants to isolate the contribution of each component: one without IR, another without CoT-R, and a version without both IR and CoT-R. The full version of *RulePilot* incorporates both IR guidance and CoT-R. We present the results of the ablation study in Figure 6. The overall trend reveals that removing either IR or CoT-R leads to a significant decrease in rule generation, and removing both components causes the most substantial drop across all metrics. Without CoT-R and IR, the model struggles to handle complex conditions and multi-step logic, leading to incomplete or logically inconsistent rules. This suggests that CoT-R and IR play a critical role in enabling the model to break down complex rule-generation tasks into manageable steps, resulting in better logical consistency and structural coherence.

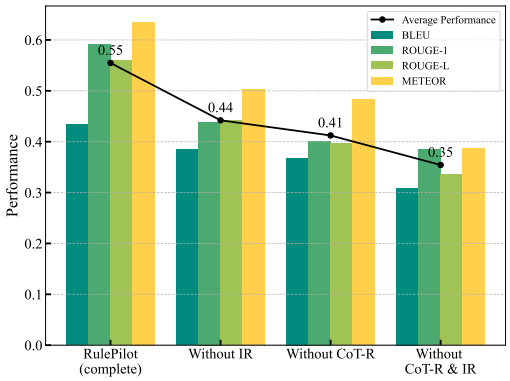


Figure 6: Ablation study on the impact of IR and CoT-R.

Additionally, to determine the specific areas influenced by IR and CoT-R, we conduct a semantic evaluation, with results shown in Figure 7. We find that removing CoT-R causes the most significant degradation in Logical Consistency (LC) and Condition Coverage (CC). Conversely, removing IR primarily affects Syntax Correctness (SC) and Readability & Maintainability (RM). The findings further highlight the complementary roles of these components, where CoT-R enhances logical structuring, and IR ensures syntactic correctness and standardization.

> **Answer to RQ3:** Both IR and CoT-R improve rule generation. CoT-R helps with logic and structuring, while IR ensures correct syntax and readability. Removing either one lowers performance, and removing both causes the biggest drop.

4.4 RQ4-Compatibility

To evaluate the compatibility of *RulePilot*, we use the dataset consisting of 30 SPL rules and their corresponding 30 KQL rules, which serve as ground truth references. These rules are sourced from

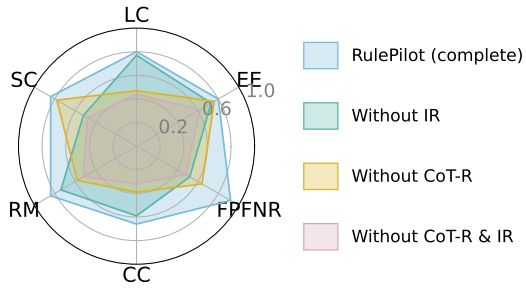


Figure 7: Ablation study for semantic-level evaluation between *RulePilot* and its variants.

real-world security applications within our industry collaborator, having been collected, segmented, and anonymized to eliminate sensitive information while preserving their syntactic and structural integrity. We further categorize the dataset into three types: aggregation-based rules, which summarize event data; list-based rules, which group multiple attributes; and join-based rules, which are complex and involve multi-source data correlation to detect cross-event security patterns. Each category contains 10 pairs of SPL and KQL rules. To maintain consistency, we collected logs oriented to Splunk, and evaluate the execution success on Splunk based on cases of converting KQL to SPL. For each rule, we convert the given KQL query into an SPL query using our model, execute it in Splunk, and compare its results with the original SPL query from the dataset. If both queries retrieve the same logs under identical conditions, the conversion is considered successful.

Evaluation Results. We present the evaluation results in Table 8 for rule conversion, categorized by rule type. The aggregation-based and list-based rules achieve perfect precision, recall and F1 (1.000), indicating that these rule types are straightforward to convert due to their simple structure and direct function mappings between SPL and KQL. Since they primarily involve statistical summarization or attribute grouping, the model can accurately translate their logic without ambiguity. However, join-based rules exhibit slightly lower performance. These rules involve multi-source data correlation, requiring careful field mapping and handling of log relationships across different event sources. The drop in performance is primarily due to boundary cases where certain event correlation logic was not fully preserved, leading to minor mismatches in retrieved log sets. Despite this, the results demonstrate that the conversion model is highly effective across different rule types, particularly for structured and statistical queries.

Table 8: Evaluation of rule conversion from KQL to SPL.

Rule Type	Precision (↑)	Recall (↑)	F1 (↑)
Aggregation-Based Rules	1.000	1.000	1.000
List-Based Rules	1.000	1.000	1.000
Join-Based Rules	0.926	0.913	0.919

This experiment demonstrates the compatibility and generalization capability of *RulePilot* for cross-SIEM rule conversion. While this experiment focuses on KQL-to-SPL translation due to log availability constraints (i.e., we collected logs oriented to Splunk SIEMs for execution success), *RulePilot* is inherently designed to support

flexible and bidirectional conversions across multiple SIEM platforms. SPL2KQL[5] is one of the publicly available rule conversion tools used in industry. Developed by Microsoft, it supports one-way translation from Splunk SPL to Microsoft Sentinel's KQL. SPL2KQL is primarily designed to ingest external detection rules into the Microsoft ecosystem and is based on traditional rule rewriting techniques such as keyword mapping, syntax tree parsing, and regex-based transformation. However, it does not support reverse conversion or semantic adaptation for other platforms. In contrast, *RulePilot* leverages LLM-based semantic understanding and an intermediate representation (IR) layer to support *bidirectional and context-aware rule conversion*, such as KQL-to-SPL, SPL-to-KQL, or even translation between other vendor formats. This flexibility makes *RulePilot* applicable to a wider range of deployment scenarios, including hybrid or transitioning security infrastructures.

To provide a more intuitive comparison when converting SPL to KQL, we select one piece of SPL from the official SPL2KQL demo repository and convert the SPL to KQL using both SPL2KQL and *RulePilot*. As shown in Figure 8, *RulePilot* can generate a semantically faithful KQL rule by aligning query operators (e.g., `contains`, `project-rename`) and adapting field references such as `TimeGenerated`, reflecting a deep understanding of both source and target semantics. In contrast, SPL2KQL applies literal keyword mappings (e.g., `TargetImage = lsass.exe`) and syntactic transformations (e.g., `rename`) without semantic reinterpretation, resulting in inaccurate or even invalid KQL logic in practical use.

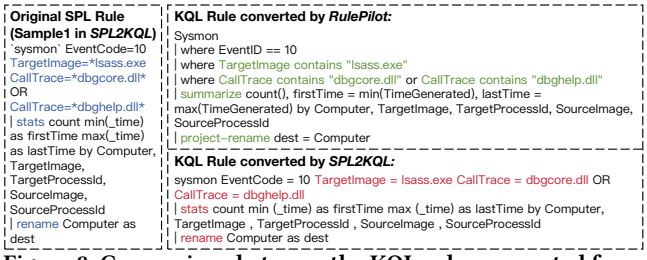


Figure 8: Comparison between the KQL rules converted from SPL via SPL2KQL and *RulePilot*.

> **Answer to RQ4:** *RulePilot* effectively supports rule conversion between Splunk SPL to Microsoft KQL, supporting the abilities of translating multiple types of rules across SIEM systems.

4.5 Case Study

We perform a case study to compare the statistical labor reduction using *RulePilot*, assessing how users of different security expertise levels perform in rule authoring with and without its support. We recruit **general users** without any background of SIEM environments and the **junior analysts** with beginner experience with SIEM exposure, under the premise that *RulePilot* incorporates expert-level expertise. We evaluate the time taken (Time used to produce a complete rule), final rule output, syntax validity (whether the rule passes vendor-side syntax checks, e.g., Splunk), and logical alignment (whether the rule logic matches the input as judged by an expert). The details are shown in our user study in [6].

The comparison study show that *RulePilot* can significantly improve the manul rule generation process for both both general users and junior analysts, reducing the time required and improving rule quality in terms of syntactic validity and logical alignment with expert-level standards.

5 Related Works

Constraint Generation. Recent studies have utilized LLMs to generate constraint logic rules in various domains [25, 26, 51]. For instance, LLMs have been applied to formal verification tasks in smart contracts [24], and to the automated extraction of generic-signature detection rule candidates from textual and visual open-source cyber threat intelligence data [36]. Additionally, LLMs have been explored for log-based anomaly detection [29], demonstrating the potential of LLMs in leveraging pre-trained knowledge to extract structured insights from large-scale log data and assist in constraint generation. However, challenges persist in modeling and capturing the intricate structures of SIEM rules, hindering the direct application of these methods to generate executable security rules.
Log Analysis. Previous works largely employ LLMs to automate log analysis [23, 29**?**], including log parsing and anomaly detection. For instance, LLM-based approaches achieve high precision in log template extraction [50] and automatic logging statement generation [49], significantly reducing manual effort. These approaches may provide valuable foundations for our work by improving log parsing and structured analysis. Unlike prior studies that focus on general log processing, our work builds upon existing SIEM rules and leverages LLMs to analyze logs and detect anomalies.

6 Discussion

Automation Level. *RulePilot* achieves a half-automated approach to SIEM-specific rule generation by embedding the logic and expertise of senior analysts. It simulates their decision-making process, including pipeline breakdown, formal template structuring, and iterative refinement. However, in practice, certain field validations and the final results require human oversights, which junior experts can handle to ensure functional-correctness and reliability.
Future Work. *RulePilot* supports query-based SIEM systems (i.e., Splunk SPL and Microsoft KQL), and can tailor to KQL generation by handling syntax differences in keyword structures and field referencing conventions. For more SIEMs, we believe that it needs a new Intermediate Representation design. *RulePilot* currently exhibits a gap when applied to non-SIEM environments, which require additional log ingestion and custom detection execution engines. We consider the extension to KQL, other SIEM, and non-SIEM environments as future works. Additionally, we aim to streamline *RulePilot* by eliminating redundant reasoning steps and improving processing speed without compromising detection quality.

7 Conclusion

In this paper, we propose *RulePilot*, an LLM-based agent system designed to automate rule creation and conversion for SIEM detection.

[5] https://azure.github.io/spl2kql/dist/index.html

[6] https://sites.google.com/view/rulepilot/user-study.

By leveraging the novel SIEM-specific intermediate representation, *RulePilot* abstracts the complexity of rule configurations into a structured and standardized format. We conduct a comprehensive evaluation of *RulePilot*, demonstrating that it can produce high-fidelity, executable SIEM-specific rules. Our case study with industry collaborators shows that *RulePilot* significantly assists general users and junior analysts by reducing rule generation time and improving rule quality, allowing them to create detection logic using natural language instead of manually adhering to strict grammar rules.

Acknowledgments

We thank the anonymous meta review and all anonymous reviewers for their insightful comments to improve this paper. This paper is supported by NUS-NCS Joint Laboratory for Cyber Security.

References

[1] 2022. Atomic Red Team™ is a library of tests mapped to the MITRE ATT&CK framework. Security teams can use Atomic Red Team to quickly, portably, and reproducibly test their environments. https://github.com/redcanaryco/atomic-red-team/tree/master/atomics.

[2] 2022. meta-llama/Llama-3.1-405B · Hugging Face — huggingface.co. https://huggingface.co/meta-llama/Llama-3.1-405B.

[3] 2022. text-embedding-ada-002. https://platform.openai.com/docs/guides/embeddings/what-are-embeddings.

[4] 2023. SolarWinds hack explained: Everything you need to know.

[5] 2024. deepseek-ai/DeepSeek-V3 · Hugging Face — huggingface.co. https://huggingface.co/deepseek-ai/DeepSeek-V3.

[6] Admin. 2025. Top 10 Soft Skills for SOC Analysts.

[7] austinmccollum. [n. d.]. Migrate Splunk detection rules to Microsoft Sentinel - Microsoft Sentinel — learn.microsoft.com. https://learn.microsoft.com/en-us/azure/sentinel/migration-splunk-detection-rules.

[8] Babak Amin Azad, Pierre Laperdrix, and Nick Nikiforakis. 2019. Less is More: Quantifying the Security Benefits of Debloating Web Applications. In *28th USENIX Security Symposium (USENIX Security 19)*. USENIX Association, Santa Clara, CA, 1697–1714. https://www.usenix.org/conference/usenixsecurity19/presentation/azad

[9] Satanjeev Banerjee and Alon Lavie. 2005. 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*. 65–72.

[10] Pedro Bernardo, Lorenzo Veronese, Valentino Dalla Valle, Stefano Calzavara, Marco Squarcina, Pedro Adão, and Matteo Maffei. 2024. Web Platform Threats: Automated Detection of Web Security Issues With WPT. In *33rd USENIX Security Symposium (USENIX Security 24)*. USENIX Association, Philadelphia, PA, 757–774. https://www.usenix.org/conference/usenixsecurity24/presentation/bernardo

[11] Sandeep N. Bhatt, Pratyusa K. Manadhata, and Loai Zomlot. 2014. The Operational Role of Security Information and Event Management Systems. *IEEE Secur. Priv.* 12, 5 (2014), 35–41. https://doi.org/10.1109/MSP.2014.103

[12] Tom Burt. 2020. Microsoft report shows increasing sophistication of cyber threats. https://blogs.microsoft.com/on-the-issues/2020/09/29/microsoft-digital-defense-report-cyber-threats/.

[13] Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2023. Teaching Large Language Models to Self-Debug. arXiv:2304.05128 [cs.CL] https://arxiv.org/abs/2304.05128

[14] The MITRE Corporation. 2024. The ATT&CK knowledge base is used as a foundation for the development of specific threat models and methodologies in the private sector, in government, and in the cybersecurity product and service community. https://attack.mitre.org/.

[15] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020 (Findings of ACL, Vol. EMNLP 2020)*, Trevor Cohn, Yulan He, and Yang Liu (Eds.). Association for Computational Linguistics, 1536–1547. https://doi.org/10.18653/V1/2020.FINDINGS-EMNLP.139

[16] IBM. 2024. IBM Security QRadar SIEM. https://www.ibm.com/products/qradar-siem

[17] Cohen's Kappa. [n. d.]. A measure of agreement between two dependent categorical samples. https://datatab.net/tutorial/cohens-kappa.

[18] Leon Kersten. 2025. A Test Tool to Evaluate the Skill Sets. In *Workshop on SOC Operations and Construction (WOSOC 2025)*. San Diego, CA, USA. https://www.ndss-symposium.org/wp-content/uploads/wosoc25-final1.pdf

[19] Patrick S. H. Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (Eds.). https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html

[20] Haitao Li, Qian Dong, Junjie Chen, Huixue Su, Yujia Zhou, Qingyao Ai, Ziyi Ye, and Yiqun Liu. 2024. LLMs-as-Judges: A Comprehensive Survey on LLM-based Evaluation Methods. arXiv:2412.05579 [cs.CL] https://arxiv.org/abs/2412.05579

[21] Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Mishig Davaadorj, Joel Lamy-Poirier, João Monteiro, Oleh Shliazhko, Nicolas Gontier, Nicholas Meade, Armel Zebaze, Ming-Ho Yee, Logesh Kumar Umapathi, Jian Zhu, Benjamin Lipkin, Muhtasham Oblokulov, Zhiruo Wang, Rudra Murthy V, Jason T. Stillerman, Siva Sankalp Patel, Dmitry Abulkhanov, Marco Zocca, Manan Dey, Zhihan Zhang, Nour Fahmy, Urvashi Bhattacharyya, Wenhao Yu, Swayam Singh, Sasha Luccioni, Paulo Villegas, Maxim Kunakov, Fedor Zhdanov, Manuel Romero, Tony Lee, Nadav Timor, Jennifer Ding, Claire Schlesinger, Hailey Schoelkopf, Jan Ebert, Tri Dao, Mayank Mishra, Alex Gu, Jennifer Robinson, Carolyn Jane Anderson, Brendan Dolan-Gavitt, Danish Contractor, Siva Reddy, Daniel Fried, Dzmitry Bahdanau, Yacine Jernite, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. 2023. StarCoder: may the source be with you! *Trans. Mach. Learn. Res.* 2023 (2023). https://openreview.net/forum?id=KoFOg41haE

[22] Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*. 74–81.

[23] Yilun Liu, Shimin Tao, Weibin Meng, Jingyu Wang, Wenbing Ma, Yuhang Chen, Yanqing Zhao, Hao Yang, and Yanfei Jiang. 2024. Interpretable online log analysis using large language models with prompt strategies. In *Proceedings of the 32nd IEEE/ACM International Conference on Program Comprehension*. 35–46.

[24] Ye Liu, Yue Xue, Daoyuan Wu, Yuqiang Sun, Yi Li, Miaolei Shi, and Yang Liu. 2024. PropertyGPT: LLM-driven Formal Verification of Smart Contracts through Retrieval-Augmented Property Generation. *CoRR* abs/2405.02580 (2024). https://doi.org/10.48550/ARXIV.2405.02580 arXiv:2405.02580

[25] Zhengxiong Luo, Qingpeng Du, Yujue Wang, Abhik Roychoudhury, and Yu Jiang. 2025. Enhancing Protocol Fuzzing via Diverse Seed Corpus Generation. *IEEE Transactions on Software Engineering* 51, 9 (2025), 2693–2709. https://doi.org/10.1109/TSE.2025.3595396

[26] Zhengxiong Luo, Huan Zhao, Dylan Wolff, Cristian Cadar, and Abhik Roychoudhury. 2026. Agentic Concolic Execution. In *2026 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 1–19.

[27] Microsoft. 2025. Microsoft Sentinel - Cloud-native SIEM Solution | Microsoft Azure — azure.microsoft.com. https://azure.microsoft.com/en-us/products/microsoft-sentinel.

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

[29] Jiaxing Qi, Shaohan Huang, Zhongzhi Luan, Shu Yang, Carol Fung, Hailong Yang, Depei Qian, Jing Shang, Zhiwen Xiao, and Zhihui Wu. 2023. Loggpt: Exploring chatgpt for log-based anomaly detection. In *2023 IEEE International Conference on High Performance Computing & Communications, Data Science & Systems, Smart City & Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys)*. IEEE, 273–280.

[30] Edward Raff, Richard Zak, Gary Lopez Munoz, William Fleming, Hyrum S Anderson, Bobby Filar, Charles Nicholas, and James Holt. 2020. Automatic yara rule generation using biclustering. In *Proceedings of the 13th ACM Workshop on Artificial Intelligence and Security*. 71–82.

[31] Stephen Robertson and Hugo Zaragoza. 2009. The Probabilistic Relevance Framework: BM25 and Beyond. *Foundations and Trends in Information Retrieval* 3 (01 2009), 333–389. https://doi.org/10.1561/1500000019

[32] RulePilot. 2025. RulePilot - Dataset — sites.google.com. https://sites.google.com/view/rulepilot/dataset.

[33] Stuart J Russell and Peter Norvig. 2016. *Artificial intelligence: a modern approach*. Pearson.

[34] Joshua Saxe. 2020. YaraML. https://github.com/sophos-ai/yaraml_rules/.

[35] Yuval Schwartz, Lavi Ben-Shimol, Dudu Mimran, Yuval Elovici, and Asaf Shabtai. 2024. LLMCloudHunter: Harnessing LLMs for Automated Extraction of Detection Rules from Cloud-Based CTI. *CoRR* abs/2407.05194 (2024). https://doi.org/10.48550/ARXIV.2407.05194 arXiv:2407.05194

[36] Yuval Schwartz, Lavi Benshimol, Dudu Mimran, Yuval Elovici, and Asaf Shabtai. 2024. Llmcloudhunter: Harnessing llms for automated extraction of detection rules from cloud-based cti. *arXiv preprint arXiv:2407.05194* (2024).

[37] Liang Shi, Zhengju Tang, and Zhi Yang. 2024. A Survey on Employing Large Language Models for Text-to-SQL Tasks. *CoRR* abs/2407.15186 (2024). https://doi.org/10.48550/ARXIV.2407.15186 arXiv:2407.15186

[38] Splunk. 2025. splunk-sdk-python/splunklib at master. https://github.com/splunk/splunk-sdk-python/tree/master/splunklib.

[39] Splunk cisco company [n. d.]. https://shorturl.at/dgTsP. State of Security 2024: The Race to Harness AI.

[40] Splunk Open-sourced Rules [n. d.]. https://github.com/splunk/security_content/tree/develop/detections/network. Splunk Open-sourced Rules.

[41] Splunk Threat Research Team [n. d.]. https://research.splunk.com/detections/. Splunk-Customized Detection Rules.

[42] Alexey Svyatkovskiy, Shao Kun Deng, Shengyu Fu, and Neel Sundaresan. 2020. IntelliCode compose: code generation using transformer. In *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020*, Prem Devanbu, Myra B. Cohen, and Thomas Zimmermann (Eds.). ACM, 1433–1443. https://doi.org/10.1145/3368089.3417058

[43] Runchu Tian, Yining Ye, Yujia Qin, Xin Cong, Yankai Lin, Yinxu Pan, Yesai Wu, Haotian Hui, Weichuan Liu, Zhiyuan Liu, and Maosong Sun. 2024. DebugBench: Evaluating Debugging Capability of Large Language Models. arXiv:2401.04621 [cs.SE] https://arxiv.org/abs/2401.04621

[44] PeiYu Tseng, ZihDwo Yeh, Xushu Dai, and Peng Liu. 2024. Using LLMs to Automate Threat Intelligence Analysis Workflows in Security Operation Centers. https://api.semanticscholar.org/CorpusID:271270843

[45] Rafael Uetz, Marco Herzog, Louis Hackländer, Simon Schwarz, and Martin Henze. 2023. You Cannot Escape Me: Detecting Evasions of SIEM Rules in Enterprise Networks. *CoRR* abs/2311.10197 (2023). https://doi.org/10.48550/ARXIV.2311.10197 arXiv:2311.10197

[46] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. 2024. A survey on large language model based autonomous agents. *Frontiers of Computer Science* 18, 6 (2024), 186345.

[47] Yue Wang, Weishi Wang, Shafiq R. Joty, and Steven C. H. Hoi. 2021. CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (Eds.). Association for Computational Linguistics, 8696–8708. https://doi.org/10.18653/V1/2021.EMNLP-MAIN.685

[48] Danning Xie, Zhuo Zhang, Nan Jiang, Xiangzhe Xu, Lin Tan, and Xiangyu Zhang. 2024. ReSym: Harnessing LLMs to Recover Variable and Data Structure Symbols from Stripped Binaries. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18, 2024*, Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie (Eds.). ACM, 4554–4568. https://doi.org/10.1145/3658644.3670340

[49] Junjielong Xu, Ziang Cui, Yuan Zhao, Xu Zhang, Shilin He, Pinjia He, Liqun Li, Yu Kang, Qingwei Lin, Yingnong Dang, et al. 2024. Unilog: Automatic logging via llm and in-context learning. In *Proceedings of the 46th ieee/acm international conference on software engineering*. 1–12.

[50] Junjielong Xu, Ruichun Yang, Yintong Huo, Chengyu Zhang, and Pinjia He. 2024. Divlog: Log parsing with prompt enhanced in-context learning. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 1–12.

[51] Ming Xu, Chuanwang Wang, Jitao Yu, Junjie Zhang, Kai Zhang, and Weili Han. [n. d.]. Chunk-Level Password Guessing: Towards Modeling Refined Password Composition Representations. In *Proceedings of 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS'21), Virtual Event, Republic of Korea, November 15 - 19, 2021*. 5–20. https://doi.org/10.1145/3460120.3484743

[52] Zhuo Zhang, Wei You, Guanhong Tao, Guannan Wei, Yonghwi Kwon, and Xiangyu Zhang. 2019. BDA: practical dependence analysis for binary executables by unbiased whole-program path sampling and per-path abstract interpretation. *Proc. ACM Program. Lang.* 3, OOPSLA (2019), 137:1–137:31. https://doi.org/10.1145/3360563

[53] Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, and Ed Chi. 2023. Least-to-Most Prompting Enables Complex Reasoning in Large Language Models. arXiv:2205.10625 [cs.AI] https://arxiv.org/abs/2205.10625