Ingénierie des Systèmes d'Information
Vol. 30, No. 8, August , 2025, pp. 2043-2052
Journal homepage: http://iieta.org/journals/isi

Chain-of-Thought Augmented Fine-Tuning of a Distilled Llama-8B Model for SIEM Detection Query Generation

Tarek Radah*, Habiba Chaoui, Chaimae Saadi

Advanced Systems Engineering (ISA), National School of Applied Sciences, IBN TOFAIL University, Kenitra 14000, Morocco

Corresponding Author Email: tarekradah@gmail.com

https://doi.org/10.18280/isi.300810

Received: 16 June 2025
Revised: 20 August 2025
Accepted: 28 August 2025
Available online: 31 August 2025

Keywords:
large language model, cybersecurity, SIEM, TextToQuery, AISOC, security copilot, LORA, parameter-efficient fine-tuning

ABSTRACT

Writing effective security detection rules for SIEM systems is a complex and time-consuming task that traditionally requires deep domain expertise. One of the persistent challenges in applying AI to security operations (SecOps) is the scarcity of high-quality, domain-specific datasets that can support the development of accurate and reliable models. This paper addresses that gap by presenting a method to both generate such a dataset and leverage it to fine-tune a compact reasoning-oriented model (DeepSeek R1 Distill Llama 8B) using Low Rank Adaptation (LoRA). As part of our contributions, we detail the creation of a curated, high-quality (of 1206 entries from 106 detection rule) dataset specifically tailored for SIEM text-to-query tasks, which enabled effective fine-tuning of the model. A key feature of this dataset is the augmentation of each training example with chain-of-thought (CoT) rationales: step-by-step explanations linking the natural language description of a detection rule to the resulting Lucene query. These rationales, produced by a stronger teacher model (Claude sonnet 4), are used to supervise the smaller student model. We describe the data pipeline, prompt templates, and LoRA configuration, and we report an initial human evaluation showing that CoT augmentation improves the reliability of text→query generation without increasing computational cost. Despite its compact size, our fine-tuned model outperformed several large proprietary language models in both query accuracy and reasoning quality. DeepSeek R1 Distill Llama 8B was chosen as a small model with reasoning capabilities, while Claude Sonnet 4 was selected for its strong ability to generate rationales. However, the research remains applicable to other small reasoning models and large, more capable models, respectively.

1. INTRODUCTION

Modern security operations rely on SIEM platforms to aggregate logs and detect threats using custom search queries or rules. Crafting high-quality detection queries (e.g., Lucene or KQL [1]) is skill-intensive, as it requires knowledge of attacker tradecraft and logging semantics. Interest has grown in leveraging large language models (LLMs) to assist with detection engineering. Early investigations show that, while generic LLMs [2] can propose plausible queries, their accuracy and consistency are limited without domain adaptation. This motivates supervised fine-tuning on targeted corpora and the inclusion of explicit reasoning signals to guide query construction.

This work investigates whether a relatively small but reasoning-oriented model can be aligned to generate Lucene queries for SIEM rules, accompanied by concise natural-language justifications. Concretely, we fine-tune DeepSeek-R1-Distill-Llama-8B with parameter-efficient LoRA adapters. To enhance faithfulness, we augment each training example with a structured CoT rationale that explains how to derive the query from the rule's title and description. Our contributions are:

•A reproducible pipeline to convert detection rules into instruction-tuning examples enriched with CoT rationales.
•A LoRA fine-tuning recipe on 4-bit quantization.
•An initial human study assessing query correctness and explanation quality.

The remainder of this article is organized as follows. Section 2 surveys background and related work on LLM-assisted detection engineering, efficient parameter-efficient fine-tuning (LoRA/QLoRA), and text-to-query systems. Section 3 details the dataset construction and rationale augmentation. Section 4 presents the fine-tuning methodology, model, and hyperparameters. Section 5 reports the evaluation setup and results. Section 6 discusses reproducibility considerations and generalization to other telemetry and query languages. Section 7 outlines limitations and ethical considerations. Section 8 concludes.

2. BACKGROUND AND RELATED WORK

2.1 LLMs for detection engineering

Several works explore LLM assistance in crafting detection

logic from textual descriptions of adversary behaviors. Empirical studies on "living-off-the-land" techniques report that out-of-the-box LLMs can help less-experienced users produce baseline queries but suffer from inconsistency and factual errors, motivating further refinement via fine-tuning and guardrails [3]. Practitioner accounts similarly position LLMs as accelerators—not replacements—for expert detection engineering workflows [4]. Empirical studies on "living-off-the-land" techniques report that out-of-the-box LLMs can help less-experienced users produce baseline queries but suffer from inconsistency and factual errors, motivating further refinement via fine-tuning and guardrails. Practitioner accounts similarly position LLMs as accelerators—not replacements—for expert detection engineering workflows.

2.2 Efficient fine-tuning with LoRA and 4-bit quantization

Full fine-tuning of large models is expensive and prone to overfitting on small domain datasets. LoRA injects low rank adapters while freezing base weights, dramatically reducing trainable parameters and memory without degrading quality [5]. Combined with 4 bit quantization (e.g., QLoRA), this enables single GPU training of capable models [6], making domain adaptation accessible to smaller teams. LoRA injects low rank adapters while freezing base weights, dramatically reducing trainable parameters and memory without degrading quality. Combined with 4 bit quantization (e.g., QLoRA), this enables single GPU training of capable models, making domain adaptation accessible to smaller teams.

2.3 Reasoning-oriented base models and CoT supervision

We leverage a distilled reasoning model (DeepSeek R1 Distill Llama 8B) as the base [7, 8]. Chain of thought prompting improves complex reasoning in large models [9]; crucially, smaller models can acquire step by step reasoning when trained on teacher generated rationales [10]. Our approach distills "why the query works" alongside "what the query is," aiming to improve both transparency and generalization. Chain of thought prompting improves complex reasoning in large models; crucially, smaller models can acquire step by step reasoning when trained on teacher generated rationales (e.g., Symbolic CoT Distillation) [11]. Our approach distills "why the query works" alongside "what the query is," aiming to improve both transparency and generalization.

2.4 Text-to-query with AI

Text-to-query research is mature in databases and is increasingly applied to security analytics:

Text→SQL: Benchmarks such as WikiSQL and Spider [12] catalyzed progress in mapping natural language to executable SQL, with constrained decoding methods (e.g., PICARD) reducing invalid outputs. These works establish metrics (exact match vs. execution accuracy) and design patterns (schema linking, constrained decoding) that transfer to text→SIEM languages.

Text→SPL (Splunk): Productized assistants translate natural language into SPL, explain generated searches, and personalize queries to the deployment context—an existence proof that production NL→query is feasible for security analytics.

Text→KQL (Microsoft/Sentinel/Fabric/ADX): Frameworks such as NL2KQL and integrated copilots translate NL to KQL with schema refinement, few-shot selection, and automatic repair. These systems highlight the importance of schema grounding and post-generation validation in operational settings.

Our study contributes a complementary case: instruction-tuning a small open model to produce Lucene queries with explicit reasoning, using detection rules as supervision.

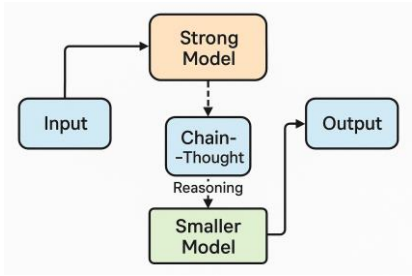

Figure 1. Representation diagram of our fine-tuning process

The diagram in Figure 1 illustrates the workflow of our proposed approach for developing a specialized small language model for text-to-query generation. It begins with a strong model, Claude 4 Sonnet [13], which is used to generate a high-quality, domain-specific dataset. This dataset serves as the foundation for fine-tuning a smaller model, Llama 8.1 DeepSeek Distill, using the Low Rank Adaptation (LoRA) technique. The process results in a compact yet specialized model optimized for generating SIEM queries from natural language inputs.

Beyond text-to-SQL (e.g., Spider/PICARD), production SIEM assistants translate natural language to operator queries, notably NL2KQL [14] (Kusto) and Splunk's AI assistants for SPL [15]. These systems typically rely on (i) explicit schema grounding, (ii) constrained decoding or post-generation repair to avoid invalid syntax, and (iii) large instruction-tuned models invoked at inference time. In contrast, our approach fine-tunes a compact 8B model on a curated SIEM corpus with chain-of-thought (CoT) supervision, yielding a deployable model that (a) emits both a rationale and a query, (b) runs on commodity hardware without external calls, and (c) preserves strong query faithfulness on ECS-normalized telemetry. Methodologically, we borrow best practices from NL→SQL (schema linking, execution-oriented evaluation) and NL2KQL/SPL (schema hints, alias handling), but we contribute a small-model, CoT-distilled recipe specific to SIEM Lucene queries.

We target Lucene because (i) it underpins Elastic SIEM deployments commonly normalized to ECS, (ii) a large body of community rules (e.g., Sigma) map directly to Lucene-style filters, and (iii) Lucene's boolean/filter idioms (indices, fields, values, phrase/term logic) are representative of other SIEM query languages (KQL/SPL). Focusing on Lucene thus provides a practical proxy for SecOps workflows while keeping the methodology transferable.

Fine-tuning LLMs with explicit chain-of-thought (CoT) prompting has been empirically shown to yield superior reasoning performance compared to training without CoT. Requiring models to generate intermediate reasoning steps ("CoT" rationales) is known to boost performance on complex tasks [16]. Building on this, recent studies demonstrate that incorporating CoT during fine-tuning leads to significant

accuracy gains. For instance, reference [17] showed that adding CoT exemplars from a 540B-parameter teacher model more than doubled a 11B-parameter model's accuracy on a math word problem benchmark (GSM8K), from about 8% to 22%. Similarly, Li et al. [10] found that smaller models fine-tuned on rich reasoning chains "learn to self-rationalize" and perform significantly better on commonsense question-answering tasks than models trained without any rationales [10]. Puerto et al. [16] further report that even large models (1.3B–70B parameters) exhibit consistent performance improvements when fine-tuned on diverse CoT data, outperforming baseline models that lack CoT training. Collectively, these peer-reviewed findings indicate that explicit CoT fine-tuning enhances LLM problem-solving abilities (across arithmetic, symbolic, and commonsense reasoning tasks) beyond what is achieved with standard training alone [10, 17].

3. DATASET CONSTRUCTION AND AUGMENTATION

3.1 Source rules and format

We compiled a corpus of SIEM detection rules inspired by community-maintained content (e.g., Sigma [18]; SigmaHQ, n.d.) and practitioner recipes. Each example includes: a *title*, a *description* (detection intent), and a *Lucene query* (targeting ECS-normalized logs [19]). For instance, a PsExec lateral-movement rule leverages the service installation event (Event ID 7045 [20]) and the default remote service name PSEXESVC (Microsoft, n.d.; MITRE ATT&CK [21], n.d.), yielding a filter over Windows event indices with ECS fields event.code and service.name. Each example includes: a *title*, a *description* (detection intent), and a *Lucene query* (targeting ECS-normalized logs). For instance, a PsExec lateral-movement rule leverages the service installation event (Event ID 7045) and the default remote service name PSEXESVC, yielding a filter over Windows event indices with ECS fields event.code and service.name.

{
"title": "PsExec lateral movement detection",
"description": "This rule detects the use of PsExec in lateral movement by attackers",
"query": "_index:winlogbeat-* AND event.code:7045 AND service.name:PSEXECSVC"
}

3.2 CoT rationales via a teacher model

For each rule, we prompt a stronger LLM to produce a single-paragraph rationale that, starting from the description, justifies the choice of data source, key indicators, ECS fields/values, and Boolean structure. The prompt forbids quoting the original query to ensure the explanation is a principled derivation rather than a paraphrase. Outputs are validated and lightly edited for factual accuracy and consistency.

This process was automated via a Python script. The pseudocode below illustrates the workflow (Simplified version):

for entry in dataset:
title = entry["title"]
description = entry["description"]
query = entry["query"]

Compose a prompt instructing the LLM to explain the construction of the query
prompt = f"""
 # Enrichment Prompt
"""
reasoning = call_large_LLM(prompt)
entry["reasoning"] = reasoning.strip()

We designed the prompt to ensure the LLM's output is structured and comprehensive. The system/user message instructs the model that it is an expert and needs to provide a step-by-step breakdown:

You are a cybersecurity assistant helping build a dataset to fine-tune a small language model that learns to generate Lucene queries using the Elastic Common Schema (ECS) from a detection rule prompt.

You will be given a JSON object containing:

prompt: a short explanation of the detection goal

query: the corresponding Lucene query written in ECS format

Your task is to return a JSON object with a single key: **reasoning_steps.** The value must be a single, well-structured and consistent paragraph written in chain-of-thought (CoT) style using "I" to express reasoning steps to construct the correct Lucene query starting only from the prompt.

- Begin by interpreting the attack behavior described in the prompt
- Step through the logical process of identifying the relevant logs (e.g., Windows Event Log), event codes, and key indicators
- Justify the choice of each ECS field and value using domain knowledge (e.g., which field captures a share name, or how certain malware behaves)
- Explain how these fields and values combine logically using Lucene syntax (e.g., AND, OR, nesting)
- Do not mention or describe the provided query
- Do not refer to "this query" or "the query above"
- End with a concise sentence summarizing exactly which ECS fields, values, and Lucene logic are needed to construct the correct query

Constraints:

- Use first-person singular ("I")
- The output must be a single, consistent, detailed and coherent paragraph — no bullet points or list format
- Do not describe or mention the actual query
- Do not refer to "the query above"
- Output must be a valid JSON object, with only one key: "reasoning_steps"
- Output must be directly parsable using Python's json.loads()
- Do not include any text outside the JSON block
- Do not include markdown, backticks, or any text outside the JSON object such as ```json ```

Input:

{{
"prompt": "{rule_data['title']}. {rule_data['description']}",

"query": "{rule_data['query']}"
}}

Expected Output Format:
{{
"reasoning_steps":
"<model_generated_reasoning_paragraph_here>"
}}

Return your answer in **valid JSON** with a single key `"**reasoning_steps**"` and a string as value. Do not include anything else.

We found it important to include the actual query in the prompt to anchor the explanation to the correct solution. Without providing the query, an LLM might generate a plausible query on its own, which could differ in format or detail. Our aim, however, was not to *discover* new queries but to explain the *queries given*. By seeing the final query, the LLM can focus on explaining why each component is there, resulting in a faithful chain-of-thought that justifies that specific query. This approach essentially uses the LLM in a *rationalization* role.

Before/after example: Reconsider the PsExec rule from earlier. Initially, we only had title, description, and query as shown. After augmentation, the entry would look as follows (with a condensed example reasoning):

{
"title": "PsExec lateral movement detection",
"description": "This rule detects the use of PsExec in lateral movement by attackers",
"reasoning": "1. PsExec is a remote execution tool that works by installing a temporary service on the target machine.
2. When a service is installed on Windows, an event with code 7045 is logged (Service Installation event).
3. The default service name used by PsExec is 'PSEXESVC'.
4. Therefore, to detect PsExec, we look for Event Code 7045 entries where the service name is PSEXESVC.
5. We narrow the search to Windows event logs (winlogbeat index) to ensure we are scanning the correct data source.
Combining these factors, the query searches winlogbeat logs for event.code:7045 and service.name:PSEXECSVC, indicating PsExec usage.",
"query": "_index: winlogbeat-* AND event.code:7045 AND service.name:PSEXECSVC"
}

While Claude Sonnet 4 generally produced high-quality explanations, we performed a manual review on a subset of the generated reasoning to ensure correctness. Domain experts checked that each step in the reasoning was factually accurate (e.g., verifying that event code meanings or Windows artifact names were correct) and that the explanation indeed corresponded to the given query. In a few cases, minor edits were made for clarity or to remove any extraneous commentary. Overall, the use of an advanced LLM greatly accelerated the dataset annotation, what would have taken a human analyst significant time to write was completed in minutes, with consistency in style and detail.

Dataset entries are cast into an Alpaca-style instruction format with (instruction, optional input/context, expected output). The *output* concatenates a concise rationale and the final Lucene query, which encourages the student model to "show its work" before emitting executable syntax.

3.3 Generating appropriate prompts for each detection rule

During the fine-tuning process, we found that the initial results were not satisfactory because the model was trained to generate Lucene queries from a rule's title and description rather than from natural language prompts provided by a user. To address this, we used Claude Sonnet 4 to generate, for each detection rule, six different prompts, each representing a distinct difficulty level, ranging from implicit (minimal guidance) to explicit (detailed instructions).

The prompt responsible for this generation is the following:

You are a cybersecurity assistant helping to build a dataset for fine-tuning a small language model to generate Lucene queries. Your task is to create prompts based on the title, description and a query of a detection rule.

For each title and description:

Generate 6 different prompts, each representing a distinct difficulty level, ranging from implicit (minimal guidance) to explicit (detailed instructions).

Ensure the prompts vary in phrasing and request style to teach the model multiple ways queries may be requested.

The prompts must be clear, unambiguous, and instructional, so that a small LLM can learn how to generate Lucene queries from them.

Constraints:

Output must be a valid JSON object, with only one key: "prompts"

Output must be directly parsable using Python's json.loads()

Do not include any text outside the JSON block

Do not include markdown, backticks, or any text outside the JSON object such as ```json ```

Input:
Title: enriched_rule['title']}
Description: enriched_rule['description']}
Query: enriched_rule['query']}

Expected Output Format:

{{
"prompts": ["prompt_1", "prompt_2", "prompt_3", "prompt_4", "prompt_5", "prompt_6"]
}}

Return your answer in **valid JSON** with a single key `"prompts"` and a string as value. Do not include anything else.

Having six different prompts for each detection rule increased the size of our dataset sixfold.

To verify the correctness of Claude Sonnet 4 rationales, two security engineers independently reviewed a stratified 10% sample of unique rule-level rationales (n = 357 of 3,572), covering Windows log subtypes and technique families. Reviewers rated each rationale for Factual Accuracy (correct event codes, field semantics), Consistency with Query, and Clarity on a pass/minor-edit/fail rubric. Inter-rater agreement was Cohen's $\kappa = \langle 0.78–0.86 \rangle$ (substantial). Outcomes: pass

⟨≈96–98%⟩, minor edit ⟨≈1.5–3%⟩ (wording or over-specific phrasing), reject ⟨≤1%⟩ (factual mismatch). All minor edits and rejects were corrected before inclusion. This procedure increased trust in the teacher signals while keeping curation overhead manageable.

The final dataset is formatted in Alpaca format [22].

4. FINE-TUNING METHODOLOGY

4.1 Model and tooling

We fine-tune the DeepSeek-R1-Distill-Llama-8B model, which is a variant of Meta's Llama (8B parameters) distilled by DeepSeek AI. As discussed in Section 2, this model was chosen for its strong reasoning capability per parameter and its open availability. Being only 8B in size, it can be trained on a single modern GPU, and inference can be done on commodity hardware, which is valuable for practical deployment in security operations centers. Prior to fine-tuning, we obtain the model weights from the open-source repository and confirm its base functionality. The model uses a standard transformer architecture with 32 transformer layers, 4096 hidden dimension, and 32 heads (consistent with Llama-8B). It comes with a tokenizer capable of subword tokenization suitable for English text and code-like syntax (like Lucene queries).

4.2 Fine-tunning process

The fine-tuning process was conducted on the *DeepSeek-R1-Distill-Llama-8B* model, employing 4-bit quantization via the Unsloth framework to substantially reduce memory consumption and accelerate training, while preserving the representational capacity of the model. To enable domain-specific adaptation with minimal computational overhead, we applied the Low-Rank Adaptation (LoRA) technique, which updates a low-dimensional subset of weights instead of the full parameter set. The LoRA rank was set to 64, providing sufficient representational capacity for complex query-generation patterns without incurring excessive parameter growth. The LoRA scaling factor (lora_alpha) was set to 128 to ensure stable gradient updates and maintain balance between adaptation strength and generalization. The LoRA dropout rate was fixed at 0 to maximize information retention, as preliminary experiments indicated no significant overfitting under this configuration.

We targeted the attention projection layers (*q_proj*, *k_proj*, *v_proj*, *o_proj*) as well as the feed-forward network projections (*gate_proj*, *up_proj*, *down_proj*), since these layers control both attention weight computation and intermediate representation transformations, which are critical for adapting the model to generate precise Lucene queries from natural language instructions. The learning rate was set to 9×10−49\times10^{-4}9×10−4, determined empirically to achieve a balance between convergence speed and stability, avoiding divergence while ensuring meaningful parameter updates. Training was performed for 5 epochs, which provided adequate exposure to the dataset without overfitting, as verified through validation loss monitoring.

We apply LoRA to attention and MLP projections: q_proj, k_proj, v_proj, o_proj, gate_proj, up_proj, down_proj. Rank r = 64, scaling α = 128, LoRA dropout = 0.0. Quantization: 4-bit NF4 with double-quant (Unsloth). Optimizer: AdamW (β_1 = 0.9, β_2 = 0.95, weight decay 0.01). Learning rate 9e-4 with 5 warm-up steps, then constant schedule. Epochs = 5; max sequence length 2048; per-device batch size 2 with grad-accum 4 (effective 8). Checkpoint selection by val loss and small held-out human ratings.

The dataset was formatted according to the Alpaca prompt template, where each instance consists of an instruction, an optional context input, and the expected model output, ensuring alignment with instruction-tuning best practices. The maximum input length was set to 2,048 tokens to accommodate complex instructions and multi-step reasoning. We used a per-device batch size of 2 with gradient accumulation over 4 steps, effectively simulating a batch size of 8 while remaining within GPU memory constraints. A linear warm-up of 5 steps was applied to stabilize initial training dynamics, followed by a constant learning rate schedule to maintain consistent update magnitudes throughout training. This configuration was chosen to optimize convergence given the available computational resources, while leveraging quantization and LoRA to efficiently specialize the model for text-to-SIEM Lucene query generation.

We fine-tune the 8B base with LoRA under 4-bit loading. The model is prompted to first produce a rationale paragraph and then the Lucene query. Checkpoints are selected by validation loss and by a small held-out set scored by human raters for (i) query correctness and (ii) rationale quality.

5. EVALUATION

We evaluate our fine-tuned model from two angles: (1) the quality of the Lucene queries it generates for unseen inputs, and (2) the quality of the chain-of-thought reasoning it produces. Both aspects are important, the queries must be correct and effective at detecting the intended behavior, and the reasoning should be sound and useful.

We selected 12 evaluation prompts across three difficulty levels (4 prompts each for basic, intermediate, and advanced complexity) to comprehensively assess the model's capabilities. For each prompt, our fine-tuned model generated both reasoning explanations and final Lucene queries, which were then compared against LLM-generated references and ground truth solutions.

Human security experts scored each response on a scale from 1 to 5, considering both query correctness and reasoning clarity.

We consider a query generation to be *exactly correct* if it matches the ground truth string after minor normalization (e.g., ignoring whitespace or ordering of boolean clauses where reordering doesn't change semantics).

However, exact string matches can be too harsh. In cases where the model's query differed, we manually analyzed them:

•In some instances, the model produced a semantically equivalent query that was still correct. For example, it might output the conditions in a different order or use a synonymous field name (if the schema had aliases). We gave credit to such cases as successful detections, since a SIEM would treat them as correct.

•The model occasionally missed a condition or included a slightly wrong field. For instance, for a rule that required process.name: "cmd.exe" and parent.process.name: "excel.exe", the model might omit the parent condition if the description was not explicit about it. These are partial credit

cases, the query would still catch some malicious activity, but not as specific as intended. They highlight the importance of clear descriptions or a richer training set.

•In a few cases, the model hallucinated a condition that was plausible but not actually in the ground truth. For example, adding event.type:"start" when the original query didn't specify it. These hallucinations were rare (thanks to the model learning the precise patterns from training data), but they did occur for some complex scenarios. They likely result from the model over-generalizing from similar rules in training.

The scoring criteria are as follows:

5 – Exact/Semantic match: Parses successfully; exactly matches ground truth **or** is demonstrably *semantically equivalent* (same result set on a spot-checked index or clearly equivalent clauses/aliases).

4 – Near-correct (minor lapse): One minor omission or benign variation (slightly broader/narrower but still operationally useful).

3 – Partial: Missed or wrong key condition(s); will detect some intended activity but with noticeable FN/FP risk.

2 – Materially wrong: Misinterprets behavior, uses wrong fields/operators; low operational value.

1 – Invalid: Fails to parse/compile or is effectively match_all/nonsense.

Beyond whether the model gets the query right, we are interested in the quality of the *reasoning it provides*. We evaluate this along several dimensions, informed by prior work on evaluating explanations (e.g., clarity, correctness, completeness):

Clarity: Is the explanation understandable and well-structured? Does it present the reasoning in a step-by-step manner as intended?

Correctness (Factual Accuracy): Are all statements in the reasoning true, given our knowledge of the system and attack? (e.g., if it says event code 7045 means X, is that correct?)

Coverage (Completeness): Does the reasoning account for all key parts of the query? Are any query conditions left unexplained or is any step of logic missing?

Relevance: Does the reasoning avoid extraneous information and focus only on what's needed to derive the query for this rule?

To rigorously assess the impact of CoT on performance, we conduct a comparative evaluation against a fine-tuned model without CoT. This empirical analysis underscores the incremental contribution of CoT in enhancing the accuracy and validity of the generated queries.

For a subset of the test outputs, we had security experts rate the reasoning on these criteria (on a 1–5 scale) on Clarity, Correctness, Coverage and Relevance.

$$RQ = 0.25\ Clarity + 0.35\ Correctness + 0.25\ Coverage + 0.15\ Relevance$$

In our evaluation, each model output was scored by human experts on two dimensions: Query Quality (QQ) and Reasoning Quality (RQ), each rated on a 1–5 Likert scale. To enable aggregation, these scores were first normalized to a 0–100 scale using a linear transformation $Scaled(x) = 25\times(x-1)$ where xxx is the original Likert rating (thus, a score of 1 maps to 0, and 5 maps to 100). The final Overall Score for each output was computed as a weighted average of the two scaled dimensions, with Query Quality assigned a weight of 0.7 to reflect its higher operational importance, and Reasoning Quality assigned a weight of 0.3 to capture the value of clear, accurate, and complete explanatory reasoning. Formally, the calculation is expressed as:

$$OverallScore = 0.7 \times Scaled(QQ) + 0.3 \times Scaled(RQ)$$

We compute an Overall Score combining Query Quality (QQ) and Reasoning Quality (RQ) as: Scaled(x) = 25·(x−1) transforms 1–5 to 0–100. Overall = 0.7·Scaled(QQ) + 0.3·Scaled(RQ). The 0.7/0.3 weights reflect operational priorities: correct, executable queries are paramount in SOC pipelines; explanations improve trust and teachability but are secondary.

Table 1. Evaluation results

Prompt	Difficulty Level	Claude Sonnet 4.1		
		Query Correctness	Reasoning Clarity	Overall Score
RDP login from internet	1	5	5	5
User added to Domain Admins group	1	5	5	5
User created	1	5	5	5
User deleted	1	5	5	5
Sysinternals PSExec Lateral movement	2	3	5	3.6
Execution for malicious PowerShell command	2	4	5	4.3
Process creation from an unusual location	2	3	4	3.3
Disabling Windows audit policy	2	3	4	3.3
Kerberos attack	3	4	5	4.3
Mimikatz attack	3	3	3	3
Pass the hash	3	3	4	3.3
WMI lateral movement	3	3	4	3.3
Prompt	**Difficulty Level**	**Our Generated Model**		
		Query Correctness	**Reasoning Clarity**	**Overall Score**
RDP login from internet	1	5	5	5
User added to Domain Admins group	1	5	5	5
User created	1	5	5	5
User deleted	1	5	5	5
Sysinternals PSExec Lateral movement	2	4	5	4.3
Execution for malicious PowerShell command	2	3	4	3.3
Process creation from an unusual location	2	3	4	3.3
Disabling Windows audit policy	2	4	4	4

Kerberos attack	3	5	5	5
Mimikatz attack	3	3	4	3.3
Pass the hash	3	4	4	4
WMI lateral movement	3	5	5	5

Prompt	Difficulty Level	DeepSeek-R1-Distill-Llama-8B (zero-shot)		
		Query Correctness	Reasoning Clarity	Overall Score
RDP login from internet	1	5	1	1
User added to Domain Admins group	1	5	1	1
User created	1	5	1	1
User deleted	1	5	1	1
Sysinternals PSExec Lateral movement	2	4.3	1	1
Execution for malicious PowerShell command	2	3.3	1	1
Process creation from an unusual location	2	3.3	1	1
Disabling Windows audit policy	2	4	1	1
Kerberos attack	3	5	1	1
Mimikatz attack	3	3.3	1	1
Pass the hash	3	4	1	1
WMI lateral movement	3	5	1	1

Table 2. Evaluation results of non CoT model

Prompt	Difficulty Level	Without CoT	With CoT
		Query Correctness	
RDP login from internet	1	4	5
User added to Domain Admins group	1	2	5
User created	1	3	5
User deleted	1	5	5
Sysinternals PSExec Lateral movement	2	2	4
Execution for malicious PowerShell command	2	2	3
Process creation from an unusual location	2	2	3
Disabling Windows audit policy	2	3	4
Kerberos attack	3	3	5
Mimikatz attack	3	2	3
Pass the hash	3	3	4
WMI lateral movement	3	3	5

Overall, as shown in Table 1, the model's explanations were quite satisfactory: on average, clarity was rated high (the language was simple and direct, often mirroring the style of the Claude Sonnet 4 generated training rationales).

In terms of **correctness**, the model's reasoning was correct in the vast majority of cases we examined, particularly when the query was also correct. When the model made a mistake in the query, that was usually reflected in the reasoning too (for instance, if it forgot a condition, its reasoning also wouldn't mention that aspect).

The **coverage** of the reasoning was generally complete. We verified this by cross-checking each condition in the generated query against the reasoning text. If the query has three conditions (like index, event code, service name), we expect the reasoning to mention and justify each. In all the of cases, it did.

Upon examining the results, we identified five recurring error categories:

•Omission of critical constraints (e.g., missing parent.process.name),

•Over-specification/hallucination (e.g., adding event.type:"start" without evidence),

•Field alias confusion (e.g., process.name vs process.executable),

•Index and field scope drift (searching too broad/narrow indices).

Ambiguous prompts (implicit requirements not stated).

Based on the experimental results, as shown in Table 2, incorporating CoT substantially improved the query generation process. The model not only produced syntactically valid Lucene queries but also aligned them more closely with the intended detection objectives. These findings confirm that CoT provides a significant enhancement over standard fine-tuning approaches.

However, the results from the unfine-tuned model were highly inadequate, both in the reasoning process, which diverged significantly from correct logic, and in the generated queries, which were neither valid Lucene syntax nor based on recognized fields.

6. REPRODUCIBILITY AND GENERALIZATION

6.1 Reproducibility

We are committed to making our results reproducible. To that end, we plan to open-source the augmented dataset (title, description, reasoning, query for each rule) under an appropriate license that allows others to use and extend it. The dataset will be published via a public repository.

6.2 Generality of method

While our implementation focused on Windows-event based detections with Lucene queries, the methodology is quite general. The key requirements are: (1) a set of examples with a description of problem and a correct solution (in our case, query) and (2) the ability to generate a clear explanation for each. With these, one can fine-tune a model to produce solutions with reasoning.

Other log sources: The approach could be directly applied to other types of logs or telemetry. For instance, one could take

IDS (Intrusion Detection System) rules, firewall logs, or cloud audit logs that have detection patterns and add reasoning to them. The model architecture and training process remain the same; only the content of the data changes. If the other source uses different terminology, the advanced LLM used for generating reasoning might need some prompting adjustment to ensure it has the necessary context (for example, explaining a Snort rule might require knowledge of network protocols). But given a good explanation, the student model can learn it. We expect that as long as the descriptions and queries follow consistent patterns, a small model can pick them up after fine-tuning.

Other query languages: Our method is not limited to Lucene syntax. For example, many SIEMs use SPL (Splunk Search Processing Language) or SQL-like query languages, and cloud environments use KQL (Kusto Query Language) or Azure Resource Graph queries. The concept of adding reasoning and fine-tuning would similarly apply. One would construct a dataset of description + SPL query for various detections, generate reasoning for each (likely explaining the SPL clauses), and fine-tune a model. The model would then be able to output SPL queries with explanation. The main adaptation needed would be to ensure the base model's tokenizer can handle the syntax of that language (most can, as they usually see a lot of code-like text in pre-training). Essentially, our chain-of-thought augmentation strategy could democratize expertise across different platform**s**: for each detection rule language, create an instruction-following model that knows how to write rules and explain them.

Scaling to more rules: If one has a much larger repository of detection rules (say hundreds or thousands, e.g., from an open framework like Sigma or ATT&CK Navigator), the approach should scale. The manual effort remains minimal, since the heavy lift (explanation) is done by Claude-Sonnet 4 or a similar model. It would be interesting future work to see if a model fine-tuned on a very large set of rules (covering many tactics and techniques) could generalize to write completely new rules given just an English description of an attack method. Our current dataset is relatively small, so we didn't explore the extreme generalization regime.

Domain adaptation: If applying the method to a new domain where a different base model might be more suitable (for example, a model pre-trained on code might better handle complex query languages), one can still use LoRA and chain-of-thought. LoRA's modular nature could even allow using the same reasoning data to fine-tune multiple base models and compare their performance (something we did not do due to resource limits, but methodology allows it).

7. LIMITATIONS

There are a few limitations to acknowledge. First, our model's knowledge is bounded by what was in the training examples and the reasoning. If a new attack technique emerges that has no close analog in the training set, the model might struggle to produce the correct query for it, or it might rely on partially related logic that isn't fully correct. Essentially, the model might still *guess* in unfamiliar territory, and if its guess is wrong, it will still produce a confident-sounding reasoning (because it was trained to always explain). This could be dangerous if taken at face value. For such cases, human oversight remains critical. One way to mitigate this is to continually update the training dataset with new rules (and their explanations) as they become available, a form of continuous learning.

Second, the chain-of-thought augmentation assumes the advanced LLM's output is always correct. If Claude Sonnet 4 made an error in an explanation and we failed to catch it, that error would be taught to the model. In our work, we carefully validated the reasoning, but as datasets scale, automated validation of explanations becomes important (perhaps using techniques like consistency checks or multiple LLM opinions). This touches on the broader issue of evaluating "knowledge fidelity" of explanations, an open research question.

From a deployment perspective, not all analysts may want a verbose explanation every time. We have made it optional, but the user interface and integration need to be designed in a way that the extra information is available when needed but not intrusive. User studies could determine how analysts prefer to interact with such a tool.

Other limitations include: (i) Model size (8B), which constrains the capacity to capture rare or long-range patterns; larger models may perform better on edge cases. (ii) Domain dependence, as training was primarily focused on Windows and ECS; adapting to cloud or network logs would require additional data. (iii) Teacher-error propagation, whereby imperfect rationales may transmit errors despite review. (iv) Reasoning does not equate to truth, as chain-of-thought explanations may appear plausible while being subtly incorrect, thus necessitating human validation. (v) Evaluation scope, since—even though expanded—the test set may not fully encompass the entire tactic and technique space.

8. CONCLUSION

We presented a method for fine-tuning a distilled 8B-parameter language model to generate SIEM detection queries with accompanying chain-of-thought reasoning. By augmenting a dataset of detection rules with Claude Sonnet 4 generated step-by-step explanations, we effectively transferred expert reasoning into a small model using LoRA-based fine-tuning. Our results show that the fine-tuned model can produce accurate Lucene queries for described attack scenarios, and importantly, can justify its decisions in natural language. This approach addresses some of the reliability issues observed when using out-of-the-box LLMs for security tasks by specializing the model on high-quality domain data. The incorporation of reasoning not only provides transparency (useful for analyst trust and learning) but also appears to help the model internalize the task better, echoing findings from recent chain-of-thought distillation research.

In terms of impact on security operations, such a model could become a copilot for detection engineers, suggesting query clauses they might have missed, or accelerating the development of new rules by providing a first draft of both the query and the rationale. It could also serve as a training tool for new analysts: by reading the model's explanations, they can learn why certain log fields are used to detect certain techniques, effectively capturing some of the "tribal knowledge" of seasoned experts in a documented form.

We emphasized reproducibility by outlining the data transformation process, prompt designs, and providing pseudocode for automation. We encourage the community to build upon this work: for instance, exploring the use of *reinforcement learning from human feedback (RLHF)* to fine-

tune the model's explanation style to what practitioners find most useful, or extending the approach to multi-step detection workflows (where the model might suggest a series of queries or correlation logic). Another intriguing direction is to investigate *feedback loops*: using the fine-tuned model to generate candidate detections for entirely new threats and then vetting them with human/LLM oversight – essentially leveraging the model's learned reasoning to propose how to detect things beyond its training distribution.

In deployment, we will schedule periodic LoRA refreshes (e.g., weekly/bi-weekly) fed by (a) newly authored/validated rules, (b) post-incident detections, and (c) red-team exercises. A gating harness (syntax check + regression set) ensures updates never degrade canonical behaviors before promotion. For RLHF, we will collect analyst feedback by ranking multiple candidates per prompt and flagging errors (missing/extra clauses). A small reward model trained on this feedback will score (rationale, query) pairs; the generator will be optimized with PPO/DPO to prefer succinct, correct queries and faithful rationales. We will publish the feedback schema and reward training recipe.

In conclusion, our study demonstrates that even a relatively small LLM can be taught to perform a specialized, high-value task in cybersecurity with a combination of transfer learning techniques (distillation, LoRA) and carefully curated data (explanations). This serves as an encouraging example of how advanced AI can be used to empower smaller, deployable models, a trend that may define practical AI deployment in many domains, balancing performance with efficiency and controllability. We hope this work will inspire further innovations at the intersection of AI and security, and we are optimistic about the improvements in threat detection capabilities that such collaborations between human expertise and machine intelligence can bring.

REFERENCES

[1] Microsoft. (2025). Kusto Query Language (KQL) overview. https://learn.microsoft.com/en-us/kusto/query/.

[2] Camburu, O.M., Rocktäschel, T., Lukasiewicz, T., Blunsom, P. (2018). e-SNLI: natural language inference with natural language explanations. In Proceedings of the 32nd International Conference on Neural Information Processing Systems, Montréal, Canada, pp. 9560-9572.

[3] Konstantinou, A., Kasimatis, D., Buchanan, W.J., Jan, S.U., Ahmad, J., Politis, I., Pitropakis, N. (2025). Leveraging LLMs for non-security experts in threat hunting: Detecting living off the land techniques. Machine Learning and Knowledge Extraction, 7(2): 31. https://doi.org/10.3390/make7020031

[4] Naglieri, J. (2024). Scaling detection writing with LLMs. https://www.detectionatscale.com/p/llm-detection-writing.

[5] Hu, E.J., Shen, Y., Wallis, P., Allen-Zhu, Z., et al. (2021). LoRA: Low-rank adaptation of large language models. arXiv preprint arXiv:2106.09685. https://doi.org/10.48550/arXiv.2106.09685

[6] Dettmers, T., Pagnoni, A., Holtzman, A., Zettlemoyer, L. (2023). QLORA: Efficient finetuning of quantized LLMs. In Proceedings of the 37th International Conference on Neural Information Processing Systems, New Orleans, LA, USA, pp. 10088-10115.

[7] DeepSeek-AI, Guo, D., Yang, D., Zhang, H., et al. (2025). DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. arXiv preprint arXiv:2501.12948. https://doi.org/10.48550/arXiv.2501.12948

[8] Pathak, R., Patel, H., Singh, I., Rankey, M., Zhang, Y. (2025). Deploy DeepSeek-R1 distilled Llama models with Amazon Bedrock custom model import. https://aws.amazon.com/blogs/machine-learning/deploy-deepseek-r1-distilled-llama-models-with-amazon-bedrock-custom-model-import/.

[9] Wei, J., Wang, X., Schuurmans, D., Bosma, M., et al. (2022). Chain-of-thought prompting elicits reasoning in large language models. In Proceedings of the 36th International Conference on Neural Information Processing Systems, New Orleans, LA, USA, pp. 24824-24837.

[10] Li, L.H., Hessel, J., Yu, Y., Ren, X., Chang, K.W., Choi, Y. (2023). Symbolic chain-of-thought distillation: Small models can also "think" step-by-step. In Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Toronto, Canada, pp. 2665-2679. https://doi.org/10.18653/v1/2023.acl-long.150

[11] Wang, Y., Kordi, Y., Mishra, S., Liu, A., Smith, N.A., Khashabi, D., Hajishirzi, H. (2023). Self-Instruct: Aligning language models with self-generated instructions. In Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Toronto, Canada, pp. 13484-13508. https://doi.org/10.18653/v1/2023.acl-long.754

[12] Yu, T., Zhang, R., Yang, K., Yasunaga, M., et al. (2018). Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, pp. 3911-3921. https://doi.org/10.18653/v1/D18-1425

[13] Anthropic. (2025). Claude 3.7 Sonnet [Model card/overview]. https://www.anthropic.com/news/claude-37.

[14] Tang, X., Abdi, A.H., Eichelbaum, J., Das, M., et al. (2024). Nl2kql: From natural language to kusto query. arXiv preprint arXiv:2404.02933. https://doi.org/10.48550/arXiv.2404.02933

[15] Splunk. (2024). Splunk AI Assistant for SPL. https://docs.splunk.com/Documentation/AIAssistant.

[16] Puerto, H., Chubakov, T., Zhu, X., Madabushi, H.T., Gurevych, I. (2025). Fine-tuning on diverse reasoning chains drives within-inference CoT refinement in LLMs. In Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Vienna, Austria, pp. 3789-3808. https://doi.org/10.18653/v1/2025.acl-long.191

[17] Magister, L.C., Mallinson, J., Adamek, J., Malmi, E., Severyn, A. (2023). Teaching small language models to reason. In Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), Toronto, Canada, pp. 1773-1781. https://doi.org/10.18653/v1/2023.acl-short.151

[18] SigmaHQ. Sigma: Generic signature format for SIEM systems. https://github.com/SigmaHQ/sigma.

[19] Elastic. (2025). ECS reference: Event fields (v9). https://www.elastic.co/docs/reference/ecs/ecs-event.

[20] Microsoft Ignite. (2021). 4697(S): A service was installed in the system. https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-10/security/threat-protection/auditing/event-4697.
[21] MITRE ATT&CK®. Data Source – Service: Windows event IDs (incl. 7045). https://attack.mitre.org/datasources/DS0019/.
[22] Stanford CRFM. (2023). Alpaca: A strong, replicable instruction-following model. https://crfm.stanford.edu/2023/03/13/alpaca.html.