Using LLMs as Assistants for Maintaining Rule-Based IOC Extractor Tools

KONSTANTIN MILEV, University of Twente, The Netherlands

Indicators of compromise (IOCs) are forensic artifacts, such as malicious IP addresses, URLs, file hashes, or malware names, that signal a likely system breach. Accurate detection and extraction of such indicators from open threat reports is crucial for timely defense, as delays in identifying IOCs can lead to missed opportunities to contain or mitigate a threat on time. Rapid IOC recognition enables real-time alerts, automated blocking, and faster incident response. Traditionally, open-source tools and research prototypes rely on hand-crafted regular expressions (regex) and rule-based extractors to identify IOCs in text. This constraint is problematic as human analysts face an overwhelming volume of unstructured reports. These static patterns struggle with variable threat report syntax, obfuscation (e.g., defanged URLs), and novel IOC formats.

Recent advancements in AI, especially Large Language Models (LLMs), offer powerful natural language understanding that can identify entities and relationships in text. By leveraging LLMs to suggest or adapt regex patterns based on new threat reports, we aim to increase IOC coverage and adaptability while reducing manual effort.

This paper concludes that, when augmented with gemma3:27b-generated regexes, the rule-based extractor's average recall jumps from 37.9% to 69.1% and its F1 score climbs from 41.0% to 55.3%, while precision increases modestly from 46.6% to 50.4%. By contrast, the smaller Regex-AI-Llama-3.2-1B:F16 model yielded only marginal gains (mean recall 39.2%, F1 32.6%).

These results show that larger LLMs can substantially broaden IOC coverage, yet the broader patterns they generate can introduce false positives. As a result, maintaining high extractor reliability in a dynamic threat landscape still depends on a human-in-the-loop workflow to review and refine LLM-suggested rules.

Additional Key Words and Phrases: Cybersecurity, Cyber Threat Intelligence (CTI), IOC extraction, Large-Language Model (LLM)

1 INTRODUCTION

Indicators of Compromise (IOCs) are concrete signs of a security breach – for example, file hashes, domain names, IP addresses, or registry keys associated with malware or attacks [7]. Security teams and tools rely on IOCs to detect and block threats; for instance, malware hash lists and C&C server addresses help network defenders identify malicious activity. Modern threat intelligence often appears as narrative reports or blogs containing IOCs, so extraction tools must parse unstructured text to extract and normalize those indicators. Traditional rule-based IOC extractors can achieve high accuracy on well-known formats, but they must be manually crafted and updated to handle new patterns. For example, defanged IOCs like `127[.]0[.]0[.]1` easily evade simple regex extractors, necessitating specialized patterns or post-processing to deobfuscate. Moreover, most IOC extractors can only perform a small set of defang transformations [7, 3, 2].

TScIT 43, July 4, 2025, Enschede, The Netherlands

Recent advancements in AI show promising results for full-stack extraction systems. For example, SecIE achieved over 92% F1 accuracy for entity extraction in threat reports [15]. Large Language Models (LLMs) in particular show great promise for cybersecurity: they can "identify latent attack patterns and vulnerabilities, assist in analyzing attack behaviors, predict threats, and even provide real-time defensive support" [23]. Recent LLM-based tools have successfully generated threat-hunting rules from unstructured threat intelligence – e.g., LLMCloudHunter automatically produced signature detection rules from textual and visual open-source CTI data with ∼99% accuracy for extracted IOCs [20]. These findings suggest LLMs can help convert natural-language threat data into meaningful rules. While LLMs can generate syntactically valid rules, in practice, they often miss corner-case patterns or produce overly broad—sometimes hallucinated—rules, as illustrated by this evaluation. Moreover, relying entirely on LLMs for IOC detection carries the risk of prompt-injection due to their "black-box" design [24].

However, little work addresses how to keep those regex rules up-to-date as new indicator formats and obfuscation techniques emerge. Maintaining high recall and precision over time, therefore, remains a manual and error-prone task. In this paper, we fill that gap by evaluating whether large language models can automate both the generation of new regex patterns and the refinement of existing ones.

This leads to the main research question: "To what extent can an LLM help analysts create new or refine existing regex rules that capture emerging IOC patterns from threat reports?"

To answer the main research questions, first, we tackle the following sub-questions:

RQ1. How does the LLM-based solution compare to the baseline in terms of recall and precision when generating new and improving existing IOC patterns?

RQ2. Which LLM models are better suited for the task of regex generation and refinement?

RQ3. What fraction of LLM-suggested regex rules are valid without modification, and how many edits do the remaining suggestions require?

2 RELATED WORK

A long line of work has addressed Indicator-of-Compromise (IOC) extraction from text, typically relying on handcrafted pattern rules. Early open-source extractors (e.g. Jager [18], IOCParser [9], Cacador [19], Cyobstract [4], IoC-Finder [10], iocextract [13]) apply large sets of regular expressions to free text. For example GoodFATR platform [3] integrates seven such tools and adds its own iocsearcher module, which applies regex for 41 indicator types to documents (PDF/HTML/text). These regex-based tools are generally effective at matching well-formed observables, but are brittle. As GoodFATR's analysis notes, even slight differences between equivalent regexes can greatly change what they match. In practice, analysts often

"defang" IOCs (e.g. writing `hxxp://` or `9[.]9[.]9[.]9` for safety), and newer obfuscations (Unicode homographs, mixed-case encodings, novel hash-like formats, etc.) continually appear [8]. Moreover, standards such as CVE undergo periodic syntax updates [14] and entirely new IOC types emerge, each requiring manually crafted regex patterns for reliable detection. Consequently, handwritten regex lists can rapidly become outdated as formats evolve and novel obfuscation techniques appear.

Given these limits, full-stack AI pipelines have been proposed to augment or replace pure regex. Systems such as SecIE [16] and cyobstract [4] combine pattern rules with statistical models. SecIE, for instance, uses pattern-based extraction for entities with strict formats (e.g. IP addresses, email addresses) and a learned encoder (BERT) for others. Notably, SecIE explicitly models many common obfuscated IOC forms (e.g. 82(dot)103(dot)137(dot)14", `x0x0[.]example[.]com`), which generic tools would miss. Other pipelines (TTPDrill [12], ThreatRaptor [6], LADDER [1], etc.) similarly mix regexes, dictionaries, and learned classifiers to extract IOCs, malware names, and related attack patterns (TTPs). However, even these end-to-end systems ultimately embed regex steps for raw indicator matching [3], and must be retrained as new obfuscation strategies and indicator types emerge.

Recent work has begun to leverage large language models (LLMs) to overcome some of these limitations. For unstructured threat reports, GPT-3.5/ChatGPT has been applied both to extract IoCs and to convert narrative CTI into structured formats. For example, Purba and Chu [17] use GPT-3.5 to parse CTI text and pull out IoCs, and Siracusano [21] et al. use it to generate STIX-formatted [1] threat intelligence from reports. Hu et al. [11] show that fine-tuning or prompting LLMs on security data can improve annotation and classification of threat entities [20]. However, naively prompting an LLM for IoC labeling tends to miss context: recent work by Froudakis et al. [5] finds that using an untuned GPT-4o model achieved only ~0.67 F1 in IoC extraction.

Beyond extraction, LLMs are also being used to generate detection rules from threat intel. The recent LLMCloudHunter framework [20] uses GPT-4 to produce custom rule templates (STIX) from open-source CTI. It includes an "IoC Extractor" component that prompts the LLM to scan report paragraphs for explicit IOCs (e.g., IPs, user-agent strings) and normalize any obfuscated forms. A subsequent "IoC Enhancer" then injects these discovered IOCs into rule candidates, expanding their signature conditions. In evaluation, LLMCloudHunter achieved ~99% precision/recall on IoCs in cloud-incident reports. Similarly, industry projects like SigmaGen train LLMs on existing Sigma rules so that "AI extracts relevant attack patterns from security blogs and maps them to MITRE ATT&CK" [22].

However, these LLM-enabled efforts focus on one-off rule creation rather than the continual maintenance of regex libraries [24], leaving open the question of how LLMs can assist in rule maintenance, which is the focus of this research.

[1] https://oasis-open.github.io/cti-documentation/stix/intro.html

3 METHODOLOGIES & APPROACH

3.1 Threat reports collection

For this evaluation, threat reports from a variety of sources from the past 2-3 years are considered. When collecting reports, priority was given to reports that contain a wide selection of IOCs in medium-to-large quantities. Reports were selected from the following datasets:

- (**RE1**-**RE5**) Selection of reports from the APTnotes dataset[2]
- (**RE6**-**RE10**) Trend Micro Threat Encyclopedia[3]
- (**RE11**) Google Threat Intelligence Group[4]

The complete list of used reports can be found in Appendix A.1. The reports were collected in PDF format, wherever possible, or as a plaintext extracted from the webpage.

Listing 1. Example labeled JSON data for **RE1**

```
{
  "report_name": "Report 0 - IOCs",
  "report_description": "Indicators of compromise
      from Report 0.",
  "iocs": {
    "ip4": [
      {
        "value_non_defanged": "173.239.196.66",
        "value_defanged": "173[.]239[.]196[.]66"
      },
      {
        "value_non_defanged": "194.126.178.8",
        "value_defanged": "194[.]126[.]178[.]8"
      },
      ...
    ],
    "fqdn": [
      {
        "value_non_defanged": "
            czyrqdnvpujmmjkfhhvs4knf1av02demj.oast
            .fun",
        "value_defanged": "
            czyrqdnvpujmmjkfhhvs4knf1av02demj.oast
            [.]fun"
      },
      ...
    ]
  }
}
```

Each report was reviewed manually to establish a reliable ground truth of IOCs, since a security analyst would always have the best IOC extraction quality. An analyst (the author of the paper) read through each report line by line and highlighted every instance of an IOC in both its original and defanged forms (e.g., "`173.239.196.66`" and "`173[.]239[.]196[.]66`"). These annotations were recorded in a simple tabular format, with columns for the indicator type (IP, URL, hash, etc.), the non-defanged value, and the corresponding defanged variant. Once all reports had been labeled, a script ingested this spreadsheet and generated a structured JSON file for each report, following the schema shown in

[2] https://github.com/aptnotes/data/blob/master/APTnotes.csv
[3] https://www.trendmicro.com/vinfo/us/threat-encyclopedia/malware
[4] https://cloud.google.com/blog/topics/threat-intelligence/

4: under an “iocs” object, each indicator type maps to an array of `{ "value_non_defanged": . . . , "value_defanged": . . . }` entries. This manual process ensured that the evaluation dataset captured every IOC instance—including edge cases of line-broken or unusually obfuscated indicators—so that LLM-augmented regex patterns could be tested against a truly comprehensive baseline.

The labeled (baseline) data is used to build a list of IOCs that iocsearcher fails to find with its list of patterns. This list of missed IOCs is used afterwards for building the prompts as described in 3.3.

3.2 Deploying local LLMs

To conduct this measurement, it was essential to use high-parameter LLM models. Selecting the appropriate models was part of a preliminary analysis, but initial considerations included recent popular models such as Gemma 3, DeepSeek, Qwen3, and others. Ollama 0.6.5 was used to deploy the models on the HPC cluster of our institution. A single NVIDIA A40 with 48 GB of VRAM was selected to support the resource demands of large LLMs.

3.3 LLM prompting

Large Language models are notoriously sensitive to prompt design. This is why well-established prompt engineering techniques have been applied such as few-shot learning.

3.3.1 System prompt. The system prompt provides the model with context and instructions. It ensures that the model outputs only a single, valid regular expression. The prompt explicitly instructs the model to omit any explanatory text or formatting. It is important to note that this does not affect the chain-of-thought in thinking models, which still reason internally but suppress explanatory output as instructed.

Listing 2. System prompt

```
You are a helpful assistant that only outputs a
    single, valid regular expression based on a
    list of indicators of compromise.
Apply appropriate regex syntax for masked or
    blocked parts.
Do NOT include any explanatory text or quotes -
    just the regex itself.
```

3.3.2 Instruction prompt. LLMs often struggle to infer the specifics of an IoC type solely from its name and a few examples. To address this, a textual description is provided alongside examples for each IoC type.

If iocsearcher does not have a predefined regex for a specific IOC, the **regex generation prompt** is used. This includes missed examples and a description to clarify the intended pattern.

Listing 3. Regex generation prompt

```
Task: Generate a Python-compatible regex pattern
    to extract the following IOC type from text.
Description: {description}
Examples:
{examples_section}
Constraints:
- Use raw-string syntax (r"...").
- Do not include anchors (^ or $).
```

If a regex already exists for a given IOC, its pattern, associated description, and missed examples are used in the **regex refinement prompt** to improve accuracy.

Listing 4. Regex refinement prompt

```
Task: Refine the existing regex to also match the
    missed examples.
Description: {description}
Regex: {existing_regex}
Missed Examples:
{examples_section}
Constraints:
- Use raw-string syntax (r"...").
- Do not include anchors (^ or $).
```

3.4 Automated testing

The testing process begins by applying the regex-based IOC extractor to threat reports and collecting the resulting extractions. Missed IOCs are collected using two strategies: per-report and aggregated across all reports. This dual approach allows comparisons on how the volume and diversity of input examples influence the quality and generalizability of the resulting regex. Generating rules from a single report may lead to highly specific patterns, while aggregated examples may produce more robust, general-purpose regexes.

When indicators are missed by the baseline extractor, they are collected and passed as input to the LLM for regex rule generation or refinement. Any valid regex suggestions are then integrated back into the rule set. The improved configuration is re-evaluated using the same dataset. This testing setup enables efficient experimentation across multiple large language models, prompt templates, and generation parameters.

3.5 Evaluation

The evaluation compares IOCsearcher’s extractions—first using the original regex patterns, then with the LLM-augmented rules—against a ground-truth dataset of manually labeled IOCs for each report (see 3.4). Performance is measured in two stages:

- Baseline run: iocsearcher operated using its original pattern set.
- LLM-augmented run: iocsearcher’s regex rules were updated with valid suggestions, and the reports were re-evaluated.

Based on this comparison, the system computes standard evaluation metrics—precision, recall, and F1 score—to quantify how well the tool captures the intended indicators. These metrics are defined as follows:

- Precision = TP / (TP + FP)
- Recall = TP / (TP + FN)
- F1 Score = 2 × (Precision × Recall) / (Precision + Recall)

Where:

- TP (True Positives) refers to IOCs correctly extracted.
- FP (False Positives) are incorrectly matched values.

- FN (False Negatives) are missed IOCs that appear in the ground truth but were not extracted.

4 RESULTS

4.1 Preliminary Analysis

A variety of large language models were initially considered as part of this research to identify those best suited for the task of regular expression generation and refinement. Initial considerations included recent popular models such as Gemma 3, DeepSeek-r1, and Llama 3.1, as well as more specialized models fine-tuned for regular expression generation. The goal was to isolate models that could reliably produce valid, precise, and generalizable regex patterns when provided with threat intelligence descriptions and examples.

After preliminary testing, two models were selected for focused analysis: `gemma3:27b` and `Regex-AI-Llama-3.2-1B:F16`. These models demonstrated consistent performance in generating valid regex patterns and captured a broad range of IOC formats with minimal prompt tuning.

While other models were briefly evaluated, their outputs showed lower reliability in terms of validity, had excessive hallucination, or limited syntax control, which made consistent testing infeasible. As such, only aggregated mean results from these additional models will be reported, without detailed per-model breakdowns. Additionally, some newer models, such as Llama 4 and Qwen 3.5, were excluded due to compatibility issues with the available version of Ollama at the time of evaluation.

The following sections focus primarily on the comparative performance of `gemma3:27b` and `Regex-AI-Llama-3.2-1B:F16`.

4.2 Validity of LLM-Generated Regex

The syntactic validity of regex patterns produced by LLMs varied notably across different models. Models such as `gemma3:27b` and `deepseek-r1:32b` consistently generated regex expressions with valid Python syntax throughout the evaluation. Across the three newly generated rules (*filename*, *filepath*, *malwarename*) and two refined rules (*cve*, *registrykey*), both models successfully produced syntactically valid patterns in all cases.

In contrast, more lightweight models such as `Regex-AI-Llama-3.2-1B:F16` and `llama3.1:8b` had lower reliability in maintaining syntactic correctness. Specifically, `Regex-AI-Llama-3.2-1B:F16` produced invalid patterns for the *filepath* IOC type, while llama3.1:8b failed to correctly generate a valid regex for *filepath*, *filename*. This result indicates that the fine-tuned version of the Llama model is better suited for generating syntactically valid patterns.

This discrepancy in syntactic validity highlights a trade-off between model size/capability and regex correctness. Higher parameter models (such as gemma3 and deepseek-r1) not only adhere better to the prompt's structural constraints but also generalize more effectively from examples. Smaller models, while faster and less resource-intensive, require post-processing and validation to ensure usable output.

Despite occasional failures, the majority of regexes across all models were syntactically valid. However, semantic validity—i.e., whether the generated pattern accurately captures the intended IOC format remains a separate concern, discussed in the next section (4.3).

To avoid runtime errors, syntactically invalid patterns are skipped and not used in testing.

4.2.1 LLM output polluted with unnecessary data. During testing, the outputs of several language models were found to be inconsistent and poorly structured. The placement of the regex pattern, as well as the presence or absence of code fencing, varied unpredictably between generations. Moreover, models like deepseek-r1, and especially smaller ones such as `llama3.1:8b`, frequently ignored explicit system-level instructions, such as "Do NOT include any explanatory text or quotes – just the regex itself." This often resulted in output that included extra explanations, Markdown formatting, or even commentary, which made automated parsing difficult or impossible.

Structured output formats (such as JSON) were explored as a potential mitigation; however, gemma3 and deepseek-r1 failed to produce a valid JSON despite specifying the output format as such. This may have been due to limitations in the Ollama version used at the time (0.6.5), though time constraints prevented a deeper investigation. As a result, enforcing output structure required manual post-processing or additional prompt engineering, neither of which were consistently reliable.

Notably, this issue was not observed with the selected models gemma3:27b and Regex-AI-Llama-3.2-1B:F16, both of which produced consistent, clean regex outputs without additional content.

4.2.2 Generated regex is syntactically invalid. Another recurring problem—particularly with smaller models—was the frequent generation of syntactically invalid regular expressions. These invalid patterns could not be compiled by standard regex engines and therefore failed entirely during IOC extraction.

For instance, when asked to generate a regex pattern to match file names based on the following examples:

```
'Client.py', 'SystemUpdate.lnk', 'VMSearch.sfx.exe',
'VMSearch.exe', '2.txt', '2.ps1',
'KFP.311.152.2023.pdf.lnk',
'Strategies of Ukraine.pdf.lnk', 'python.exe',
'powershell.exe', 'wody.pdf'
```

One model produced the following invalid regex:

```
.\(\[a-zA-Z0-9]+\.[a-zA-Z]{1,5}\)\.lnk?\
```

This pattern is unusable for several reasons:

(1) Illegal escape sequences - The backslashes before the parentheses `"("`, `")"`, `"["`, `"]"` suggest an attempt to escape the parentheses, but most modern regex engines (including Python's *re*) do not accept either as valid escapes. Because the prompt did not specify a particular regex engine, the model incorrectly assumed that they would be universally valid. This could be considered a missing element of the prompt.
(2) The dot at the very beginning, in its current form, is matching *any* character. It most likely should have been escaped using `\.` to match just the start of the file extension.
(3) Even if the backslashes were a valid escape, this regex would require the text to contain actual parentheses around the filename (e.g. `(example).lnk`), which none of our sample filenames include.

Table 1. Comparison of Baseline and LLM-Augmented Performance (gemma3:27b)

Report	Baseline			LLM-Augmented (Individual)			LLM-Augmented (Combined)		
	Prec.	Rec.	F1	Prec.	Rec.	F1	Prec.	Rec.	F1
Report 1 (**RE1**)	0.8246	0.5281	0.6438	0.494	0.9213	0.6431	0.4314	0.7416	0.5455
Report 2 (**RE2**)	1	0.8421	0.9143	0.8537	0.9211	0.8861	0.9174	0.8947	0.9315
Report 3 (**RE3**)	0.8636	0.5278	0.6525	0.6889	0.8611	0.7654	0.5577	0.8056	0.6591
Report 4 (**RE4**)	0.875	0.5185	0.6512	0.8	0.5926	0.6809	0.7143	0.7407	0.7273
Report 5 (**RE5**)	0.6986	0.75	0.7234	0.4265	0.8529	0.5686	0.6	0.8382	0.6994
Report 6 (**RE6**)	N/A	N/A	N/A	0.1579	0.4286	0.2308	0.1538	0.2857	0.2
Report 7 (**RE7**)	N/A	N/A	N/A	0.75	0.6923	0.72	0.1515	0.3846	0.2174
Report 8 (**RE8**)	N/A	N/A	N/A	0.1429	0.0769	0.1	0.044	0.3077	0.0769
Report 9 (**RE9**)	N/A	N/A	N/A	0.1667	0.75	0.2727	0.1429	0.25	0.1818
Report 10 (**RE10**)	N/A	N/A	N/A	0.2	0.5	0.2857	0.0833	0.25	0.1250
Report 11 (**RE11**)	0.8667	1	0.9286	0.8667	1	0.9286	0.74	0.97	0.46
Mean	**0.4662**	**0.3788**	**0.4103**	**0.5043**	**0.6906**	**0.5529**	**0.4124**	**0.5881**	**0.4385**

Table 2. Comparison of Baseline and LLM-Augmented Performance (Regex-AI-Llama-3.2-1B:F16)

Report	Baseline			LLM-Augmented (Individual)			LLM-Augmented (Combined)		
	Prec.	Rec.	F1	Prec.	Rec.	F1	Prec.	Rec.	F1
Report 1 (**RE1**)	0.8246	0.5281	0.6438	0.4845	0.5281	0.5054	0.4608	0.5281	0.4921
Report 2 (**RE2**)	1	0.8421	0.9143	1	0.8421	0.9143	0.7805	0.8421	0.8101
Report 3 (**RE3**)	0.8636	0.5278	0.6525	0.8636	0.5278	0.6552	0.5278	0.5278	0.8101
Report 4 (**RE4**)	0.875	0.5185	0.6512	0.3256	0.5185	0.4	0.4667	0.5185	0.4912
Report 5 (**RE5**)	0.6986	0.75	0.7234	0.0453	0.75	0.0857	0.4286	0.75	0.5455
Report 6 (**RE6**)	N/A	N/A	N/A	0	0	0	0	0	0
Report 7 (**RE7**)	N/A	N/A	N/A	0	0	0	0	0	0
Report 8 (**RE8**)	N/A	N/A	N/A	0	0	0	0	0	0
Report 9 (**RE9**)	N/A	N/A	N/A	0.0667	0.1429	0.0909	0	0	0
Report 10 (**RE10**)	N/A	N/A	N/A	0	0	0	0	0	0
Report 11 (**RE11**)	0.8667	1	0.9286	0.8667	1	0.9286	0.74	0.97	0.46
Mean	**0.4662**	**0.3788**	**0.4103**	**0.3320**	**0.3918**	**0.3255**	**0.3095**	**0.3760**	**0.3281**

4.3 Analysis by IOC Type

4.3.1 filename. Extracting filenames poses a unique challenge: simple patterns easily over-match non-file strings, while overly strict patterns miss valid but uncommon formats.

Single-report prompting

When fed only a few examples from one report, both models commonly produced:

```
\w+\.\w+
```

Although this captures nearly every "name.extension" combination, it also flags unrelated text segments. For instance:

- 127.0.0.1 -> false positives "127.0" and "0.1"
- www.google.com - false positive "www.google"

Aggregated-report prompting

After exposing gemma3:27b to a wide variety of filenames drawn from all reports, the generated pattern became:

```
[A-Za-z0-9._-]+(\.[A-Za-z0-9]+){1,2}
```

This pattern handles cases with hyphens and dots in the filename more effectively, allowing for more realistic file name structures found in practice (e.g., `my-file.v2.backup`). It also supports filenames with two file extensions, such as **.tar.gz**, which are common in compressed archive formats. Additionally, it provides greater flexibility in extension matching.

4.3.2 filepath. Extracting file paths reliably poses several challenges, especially on Windows systems where environment variables, nested directories, and multiple file extensions must all be handled. Two different prompting strategies illustrate these issues:

Single-report prompting

When provided only a handful of examples from one report, both `gemma3:27b` and `Regex-AI-Llama-3.2-1B:F16` tended to generate highly specific patterns such as:

```
(?:%User Temp%\\(?:[^\\]+\\.dll)|%Windows%\\regedit\.exe|
%Program Files%\\Seagull\\BarTender Suite\\bartend\.exe|
%User Temp%\\(?:[^\\]+\\.ttf)|%User Temp%\\
(?:[^\\]+\\.reg)|%User Temp%\\{[0-9a-fA-F]+\\.dll)
```

While this captures exactly the examples given, it fails to match:

- Paths in other system variables (e.g. `%APPDATA%`)
- Nested folder structures beyond those listed
- Different file extensions (e.g. .exe, .zip, .txt)
- UNC (network-share) paths (e.g. `\\SERVER\payload.exe`)

Aggregated-report prompting

After pooling dozens of file-path examples from all reports, `gemma3:27b` learned a much more general but still precise pattern:

```
(?:%[A-Z]+%|CSIDL_[A-Z]+|\\(?:[0-9]{1,3}\.){3}[0-9]
{1,3}(?:@\d{1,5})?|C:\\(?:Users\\)?(?:Public\\
Documents\\)?)[^\\]+\.(?:vbs|exe|dll|zip|txt|lnk|
bat|js|html|reg|png|gif|css|htc|log|tmp)|%User Temp%
\\(?:inH{[0-9a-fA-F]+}\\)?(?:[^\\]+\\)*[^\\]+\.
(?:vbs|exe|dll|zip|txt|lnk|bat|js|html|reg|png|gif|
css|htc|log|tmp)
```

Key improvements compared to the first pattern:

- Variable support: Matches any `%VARIABLE%` or `CSIDL_` placeholder.
- Network paths: Allows UNC paths or IP-at-port notation (`\\192.168.0.1@8080\. . .`).
- Drive letters: Optional `C:\Users\` or `C:\Public\Documents\`
- Extension list: A comprehensive set of common file types, both primary (e.g. .exe, .dll, .zip) and secondary (e.g. nested .tmp or .log).
- Nested directories: Supports zero or more additional subfolders with chained `\\. . . \.` segments.

By increasing the diversity and volume of examples, the aggregated pattern achieves a balance between generality and precision. It captures realistic Windows paths while avoiding random strings or URLs. This significantly reduced false positives on non-filepath strings (such as portions of URLs or log entries) compared to the overly narrow, single-report regex.

4.3.3 malwarename. Crafting a regex for malware identifiers is tricky because reports usually mention only a few specific families, causing even aggregated patterns to lean heavily on known prefixes. The following regex was generated from dozens of malware names pooled across all reports:

```
(?:Win32|MSIL|Trojan(?:Spy|)|HackTool|Adware)\.
(?:[A-Za-z0-9\.]+)\.(?:[A-Za-z0-9]+)
```

Limitations:

- Hard-coded families: Any malware name without one of the listed prefixes (e.g. “Conti” or “Qbot”) will be missed entirely.
- No hyphens or underscores: Names like DarkComet-RAT or NanoCore_Injector aren’t captured.
- Only three segments: Simpler names without two dots or deeper hierarchies with more than two dots may fail to match.

Despite these constraints, this aggregated pattern strikes a balance between precision and coverage for well-known families of malware, but additional refinement or a more general fallback rule would be needed to cover emerging or unconventional malware names.

4.3.4 cve. Detection of CVE identifiers is one of the simpler extraction tasks—yet even here, static regexes can lag behind evolving standards. By default, IOCsearcher uses:

```
CVE\-[0-9]{4}\-[0-9]{4,6}
```

This pattern covers CVEs issued between 1999 and 2015 (which always had between four and six digits in the second component) but fails to match newer entries that exceed six digits.

LLM-Refined Pattern Both `gemma3:27b` and `Regex-AI-Llama-3.2-1B:F16` consistently suggested:

```
CVE\-[0-9]{4}\-[0-9]{4,}
```

This pattern has an unbounded upper limit and ensures that any number of digits above four after the year is accepted. This aligns with the MITRE CVE syntax change implemented in 2016, which removed the six-digit cap on identifier numbers [14].

4.4 Model Comparison

The evaluation reveals a significant performance gap between models. This comparison focuses on `gemma3:27b` and `Regex-AI-Llama-3.2-1B:F16`, with other models providing context for the range of capabilities.

As shown in Tables 1 and 2, `gemma3:27b` consistently outperformed `Regex-AI-Llama-3.2-1B:F16`, achieving higher F1 scores across most reports, especially with the aggregated prompting strategy. Where the baseline failed to find any IOCs (**RE6**-**RE10**), `gemma3:27b` generated effective rules while `Regex-AI-Llama-3.2-1B:F16` often yielded zero scores. This highlights `gemma3:27b`’s ability to generalize from diverse examples. There was also a qualitative difference seen in its more sophisticated filename and filepath patterns.

`gemma3:27b` also proved more reliable, consistently generating valid regex and adhering to prompt instructions. In contrast, `Regex-AI-Llama-3.2-1B:F16` failed on malwarename and fqdn IOC types, showing its limitations.

While `deepseek-r1:32b` produced decent patterns, it failed to write the output in a structured way regardless of the prompt. This made it challenging to extract the regex to use for testing, and thus, this model was omitted. At the other end, `llama3.1:8b` exemplified the issues with smaller models, frequently producing invalid regex and ignoring prompt instructions, making it unsuitable for an automated workflow.

5 DISCUSSION

The results clearly indicate that an LLM-assisted workflow can substantially improve the recall of existing rule-based extractors. This finding directly addresses **RQ1**, which compares the LLM-based solution to the baseline in terms of recall and precision. As shown in Table 1, the gemma3:27b model, when prompted with missed IOCs, consistently enabled the baseline iocsearcher tool to identify indicators it had previously overlooked. This was particularly evident in the successful refinement of the cve pattern, where the LLM correctly adapted an outdated rule to a new, more flexible standard—a task that would otherwise require manual intervention by an analyst aware of the syntax change.

However, this increase in recall was frequently accompanied by a decrease in precision. The LLM-generated rules, while capturing more true positives, also introduced a higher number of false positives. This suggests that the models, especially when generalizing from a limited set of examples, tend to create broader patterns than necessary. This trade-off is a critical consideration for practical deployment. While higher recall reduces the risk of missing IOCs, low precision can overwhelm analysts with false alerts, eroding trust in

the system. The goal is not just to find more IOCs, but to find them reliably.

The comparison between per-report (individual) and aggregated (combined) prompting strategies revealed another layer of complexity. The aggregated approach, which provided the LLM with a more diverse set of examples, was hypothesized to create more robust, general-purpose rules. This held true for filename and filepath IOCs, where the aggregated patterns were qualitatively superior, handling a wider variety of formats and edge cases. Conversely, for some reports, the per-report strategy yielded a better F1 score. This suggests that for highly contextual or unique IOC formats, a specific, targeted regex may be more effective than a generalized one, highlighting the need for an analyst to choose the right strategy based on the specific maintenance task.

In addressing **RQ2**, which asks which LLM models are better suited for the task, the findings strongly favor larger, more capable models. gemma3:27b demonstrated superior performance in generating both syntactically and semantically valid regex compared to the smaller Regex-AI-Llama-3.2-1B:F16 model. This observation also helps answer **RQ3**, which questions the fraction of valid, ready-to-use regex suggestions. As noted in Section 4.2, the larger models consistently produced syntactically valid regex that required no modification for basic usability. In contrast, smaller models frequently generated invalid patterns that were unusable without significant manual correction. This implies that the fraction of usable suggestions is highly dependent on model capability.

The analysis by IOC type further illuminated the LLMs' reasoning process. For well-structured indicators like CVEs or file paths with predictable patterns, the models excelled. However, for more amorphous types like malwarename, the LLM struggled to generalize beyond the specific examples provided, producing a regex that was essentially a hard-coded list of known malware families. This underscores a key limitation: LLMs are pattern matchers, not cybersecurity experts. They cannot infer the platonic ideal of a "malware name" but can only generalize from the data they are shown.

6 LIMITATIONS

While this research offers valuable insights, its findings are subject to several limitations that may affect the generalizability and interpretation of the results.

First, **the reliability of the baseline comparison** is a key concern. The iocsearcher tool, used for the baseline, might systematically fail to read IOC patterns from PDFs, especially when an indicator is stretched across two lines. This can artificially lower the baseline's performance, meaning the perceived improvement from LLM-augmented rules could be inflated, as some "missed" IOCs might stem from parsing errors rather than inadequate regex.

Second, the study highlights the **model's sensitivity to data quality**. A single incorrectly written IOC in a report, such as a SHA256 hash containing 63 characters instead of 64, can mislead a model into attempting to "refine" an already optimal regex. This could result in a less accurate pattern. To mitigate this, the evaluation deliberately avoided refining patterns for IOC types with well-established formats (e.g., hashes, URLs, IPv4 addresses), which itself constitutes a limitation by narrowing the scope of the maintenance task studied.

Third, the study's **scope is limited** by its reliance on a curated set of eleven threat reports. This small sample, though drawn from reputable sources, may not be representative of the diverse landscape of cyber threat intelligence. The specific IOC formats and reporting styles present in this dataset mean the performance metrics and the quality of the generated regex may not generalize to other sources.

Finally, the research revealed significant **limitations in the capabilities of the tested LLMs**, particularly smaller models. These models struggled with both syntactic correctness and instruction adherence. For instance, some produced syntactically invalid patterns containing illegal escape sequences, while others ignored system prompts and polluted the output with explanatory text, complicating automated parsing.

7 FUTURE WORK

There are several approaches to build on this research:

First, expanding the evaluation to a much larger and more varied corpus of threat reports would help validate the generalizability of these findings.

Second, exploring more advanced prompt engineering, such as Chain-of-Thought (CoT) or self-correction prompts, could potentially improve the models' ability to generate more precise rules.

Third, fine-tuning smaller, open-source models specifically for the task of regex generation could offer a more efficient and reliable alternative to relying on large, general-purpose models. Finally, developing an interactive toolchain that formally integrates the human feedback loop—allowing an analyst to easily accept, reject, or edit suggestions and have those corrections inform future generations—would be a valuable next step toward practical implementation.

8 CONCLUSIONS

This research aimed to determine how Large Language Models (LLMs) can assist analysts in maintaining regex rules for Indicator of Compromise (IOC) extraction. The study confirms that LLMs can be effective assistants, but their usefulness is maximized only when supervised by a human.

LLM-assisted regex generation significantly improves recall and allows the capturing of new IOC formats, but often at the cost of precision by generating overly broad rules and false positives. Model capability is also a critical factor; larger models like `gemma3:27b` consistently produce more syntactically and semantically valid suggestions than smaller models.

In conclusion, LLMs can substantially accelerate the maintenance of rule-based IOC extractors by drafting patterns for structured IOCs like CVEs. However, because the models can struggle with generalization and are sensitive to data quality issues, human oversight is essential. The most effective approach is a "human-in-the-loop" model where an analyst validates the LLM-suggested rules. This method would combine the AI's speed and general knowledge with the security analyst's expertise to ensure that the LLM-generated suggestions are meaningful.

9 ACKNOWLEDGMENTS

I would like to thank my supervisor Zsolt Kucsván for providing me with the resources and guidance that enabled me to do this research.

REFERENCES

[1] Md Tanvirul Alam et al. "Looking Beyond IoCs: Automatically Extracting Attack Patterns from External CTI". In: *Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses*. RAID '23. Hong Kong, China: Association for Computing Machinery, 2023, pp. 92–108. ISBN: 9798400707650. DOI: 10.1145/3607199.3607208. URL: https://doi.org/10.1145/3607199.3607208.

[2] Stephen Brannon and williamgibb. *stephenbrannon/IOCextractor*. Jan. 14, 2016. URL: https://github.com/stephenbrannon/IOCextractor.

[3] Juan Caballero et al. "The Rise of GoodFATR: A Novel Accuracy Comparison Methodology for Indicator Extraction Tools". In: *Future Generation Computer Systems* 144 (2023), pp. 74–89. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2023.02.012. URL: https://www.sciencedirect.com/science/article/pii/S0167739X23000535.

[4] Carnegie Mellon University Software Engineering Institute. *CyObstract: Cyber Obfuscation Analysis Tool*. https://github.com/cmu-sei/cyobstract. Accessed: 2025-06-18. 2024.

[5] Evangelos Froudakis et al. *Uncovering Reliable Indicators: Improving IoC Extraction from Threat Reports*. 2025. eprint: arXiv:2506.11325.

[6] Peng Gao et al. *Enabling Efficient Cyber Threat Hunting With Cyber Threat Intelligence*. 2021. arXiv: 2010.13637 [cs.CR]. URL: https://arxiv.org/abs/2010.13637.

[7] Peng Gao et al. *ThreatKG: An AI-Powered System for Automated Open-Source Cyber Threat Intelligence Gathering and Management*. 2022. eprint: arXiv:2212.10388.

[8] Stefan Grimminck. *A Standard for Safe and Reversible Sharing of Malicious URLs and Indicators*. Internet-Draft draft-grimminck-safe-ioc-sharing-03. Work in Progress. Internet Engineering Task Force, Apr. 2025. 6 pp. URL: https://datatracker.ietf.org/doc/draft-grimminck-safe-ioc-sharing/03/.

[9] Alexander Hanel. *ioc_parser: A Parser for Mandiant IOC XML Files and Text-Based IOC Extraction*. https://github.com/armbues/ioc_parser. Accessed: 2025-06-18. 2013. URL: https://github.com/armbues/ioc_parser.

[10] Fred Hightower. *IOC Finder: Indicator of Compromise Extraction Tool*. https://github.com/fhightower/ioc-finder. Accessed: 2025-06-18. 2024.

[11] Yuelin Hu et al. "LLM-TIKG: Threat intelligence knowledge graph construction utilizing large language model". In: *Computers & Security* 145 (2024), p. 103999. ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2024.103999. URL: https://www.sciencedirect.com/science/article/pii/S0167404824003043.

[12] Ghaith Husari et al. "TTPDrill: Automatic and Accurate Extraction of Threat Actions from Unstructured Text of CTI Sources". In: Dec. 2017, pp. 103–115. DOI: 10.1145/3134600.3134646.

[13] InQuest. *iocextract: Indicator of Compromise Extraction Tool*. https://github.com/InQuest/iocextract. Accessed: 2025-06-18. 2024.

[14] MITRE Corporation. *CVE Identifiers: Syntax Change*. Accessed: 2025-06-18. 2013. URL: https://cve.mitre.org/cve/identifiers/syntaxchange.html.

[15] Youngja Park and Taesung Lee. "Full-Stack Information Extraction System for Cybersecurity Intelligence". In: *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing: Industry Track*. Ed. by Yunyao Li and Angeliki Lazaridou. Abu Dhabi, UAE: Association for Computational Linguistics, Dec. 2022, pp. 531–539. DOI: 10.18653/v1/2022.emnlp-industry.54. URL: https://aclanthology.org/2022.emnlp-industry.54/.

[16] Youngja Park and Taesung Lee. "Full-Stack Information Extraction System for Cybersecurity Intelligence". In: *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing: Industry Track*. Ed. by Yunyao Li and Angeliki Lazaridou. Abu Dhabi, UAE: Association for Computational Linguistics, Dec. 2022, pp. 531–539. DOI: 10.18653/v1/2022.emnlp-industry.54. URL: https://aclanthology.org/2022.emnlp-industry.54/.

[17] Moumita Das Purba and Bill Chu. "Extracting Actionable Cyber Threat Intelligence from Twitter Stream". In: *Proceedings of the 18th International Conference on Risks and Security of Internet and Systems (CRiSIS)*. Oct. 2023, pp. 1–6. DOI: 10.1109/ISI58743.2023.10297205.

[18] Scott Roberts. *Jager: Extract Indicators of Compromise from Text*. https://github.com/sroberts/jager. Accessed: 2025-06-18. 2020. URL: https://github.com/sroberts/jager.

[19] Simon Roberts. *Cacador: A Tool for Command and Control Detection*. https://github.com/sroberts/cacador. Accessed: 2025-06-18. 2024.

[20] Yuval Schwartz et al. *LLMCloudHunter: Harnessing LLMs for Automated Extraction of Detection Rules from Cloud-Based CTI*. 2024. eprint: arXiv:2407.05194.

[21] Giuseppe Siracusano et al. *Time for aCTIon: Automated Analysis of Cyber Threat Intelligence in the Wild*. 2023. arXiv: 2307.10214 [cs.CR]. URL: https://arxiv.org/abs/2307.10214.

[22] FPT Software. *SigmaGen: Automated Sigma Rule Generation –Leveraging LLMs for Continuous Rule Updates*. Presented at MITRE CTID APAC 2025. May 2025. URL: https://ctid.mitre.org/events/apac-2025/07%20-%20SigmaGen.pdf.

[23] Shuang Tian et al. *Exploring the Role of Large Language Models in Cybersecurity: A Systematic Survey*. 2025. eprint: arXiv:2504.15622.

[24] Ming Xu et al. *IntelEX: A LLM-driven Attack-level Threat Intelligence Extraction Framework*. 2024. eprint: arXiv:2412.10872.

AI STATEMENT

During the preparation of this paper, I used ChatGPT to assist in spell-checking, grammar refinement, and restructuring of sentences to better conform to academic writing standards. Additionally, I used it for a preliminary exploration of related work and to receive writing suggestions. All AI-generated outputs were carefully reviewed before being implemented in the research. I take full responsibility for the contents and conclusions presented in this work.

A APPENDICES

A.1 Appendix A.1

Table 3. Threat report URLs

ID	URL
RE1	https://app.box.com/s/s2uqsgl0krjgy2q806x3vy0hx6jfjem8
RE2	https://app.box.com/s/dd69xw7nerwy3w6zkqapfwhrsacgtxxj
RE3	https://app.box.com/s/rel585pqa8fc80voc96wt9oidsik0pvl
RE4	https://app.box.com/s/hn3cttmgg2ijnz0qr41iwh6kpgenapvr
RE5	https://app.box.com/s/hwalx6d0jzl86zfotki735i40j5a7fr0
RE6	https://www.trendmicro.com/vinfo/us/threat-encyclopedia/malware/trojanspy.ps1.poweater.a
RE7	https://www.trendmicro.com/vinfo/nl/threat-encyclopedia/malware/hacktool.win32.patcher.0bdg14
RE8	https://www.trendmicro.com/vinfo/us/threat-encyclopedia/malware/adware.win32.installcore.r002c0rce25
RE9	https://www.trendmicro.com/vinfo/us/threat-encyclopedia/malware/trojan.html.powload.c
RE10	https://www.trendmicro.com/vinfo/nl/threat-encyclopedia/malware/trojan.lnk.argulong.b
RE11	https://cloud.google.com/blog/topics/threat-intelligence/coldriver-steal-documents-western-targets-ngos