Leveraging LLMs for Automated IDS Rule Generation: A Novel Methodology for Securing Industrial Environments

Manez Moreno, Xabier Sáez-de-Cámara, Aitor Urbieta
Ikerlan Technology Research Centre
Basque Research and Technology Alliance (BRTA)
Arrasate-Mondragón, Spain
{mmoreno, xsaezdecamara, aurbieta}@ikerlan.es

Mikel Iturbe
Department of Electronics and Computing
Mondragon Unibertsitatea
Arrasate-Mondragón, Spain
miturbe@mondragon.edu

Abstract: Industrial Control Systems have become increasingly connected under Industry 4.0, raising the risk of sophisticated cyber threats to critical infrastructure. Traditional Intrusion Detection Systems (IDS) that rely on manually crafted static rules struggle to adapt quickly to new attacks, while machine-learning-based detectors face challenges in interpretability and require extensive domain-specific data. This paper proposes a novel framework that leverages Large Language Models (LLMs) to automatically generate IDS rules for Suricata, addressing the gap between static rule-based security and the need for dynamic, adaptive defenses. We integrate advanced LLMs into an automated workflow for rule generation, refinement, and validation. Through experiments, the framework demonstrates that LLM-generated rules can achieve high detection accuracy and low false positive rates while optimized prompt strategies and moderate packet-level details boost rule accuracy and effectiveness. The results highlight the potential of LLMs to enhance cybersecurity in industrial environments by rapidly producing transparent and effective IDS rules.

***Index Terms*— Large Language Models (LLMs), Automated Rule Generation, Intrusion Detection Systems (IDS), Threat Detection**

Type of contribution: Original Research

I. INTRODUCTION

Modern industrial environments increasingly interconnect Operational Technology (OT) with Information Technology (IT) systems, yielding efficiencies but also exposing critical infrastructure to cyber threats. High-profile attacks like *Stuxnet* and *Triton* have shown that breaches in Industrial Control Systems (ICS) can cause physical damage and safety risks. Indeed, a recent ICS-CERT report noted "33.8% of ICS computers were attacked in the first half of 2021," [1] underscoring the severity of the threat landscape. Intrusion Detection Systems (IDS) are a cornerstone of defense in such environments, continuously monitoring network traffic for malicious patterns. However, conventional IDSs depend on static signature rules that are manually written and updated. Crafting and maintaining these rules is labor-intensive and cannot keep pace with evolving attack techniques. Although Machine Learning (ML) and anomaly-detection approaches have been explored to automate ICS threat detection, they often require large training datasets and struggle with transparency and trust – critical factors in industrial settings where explainability and determinism are valued.

Recent advances in LLMs offer a promising alternative for automating IDS rule creation. LLMs like GPT and others have shown the ability to generate structured text based on prompts, suggesting they could be used to synthesize IDS rules from descriptions of network activity. The research gap addressed by this work lies in whether and how LLMs can be harnessed to produce operationally effective IDS rules for industrial networks, bridging the agility of AI with the interpretability of expert-written signatures. In this paper, we introduce a framework that leverages state-of-the-art LLMs to automatically generate Suricata IDS rules from network traffic captures. Unlike prior approaches that either require extensive model training or produce only coarse-grained signatures, our method uses off-the-shelf LLMs guided by carefully designed prompts to produce refined, context-rich rules. We evaluate the framework on multiple realistic ICS attack scenarios to validate that the LLM-generated rules can detect attacks with high accuracy and low false positives. The novelty of our approach is in using LLMs as an automated *IDS rule writer* systematically exploring prompt engineering, model selection, and input detail granularity to optimize rule quality. Through this, we aim to reduce the manual effort in developing IDS content while maintaining the expert knowledge and transparency that rule-based systems provide.

The primary objective of this research is to develop and evaluate an automated IDS rule generation framework using LLMs for securing industrial networks. To address this objective, we pose four key research questions (*RQ*):

RQ1: Effectiveness of LLMs in IDS Rule Generation – *How effectively can LLMs generate accurate and relevant IDS rules based on industrial network traffic data?* This question examines if LLMs can produce rules that not only parse correctly but also match malicious patterns in traffic.

RQ2: Impact of Prompt Engineering – *How do different prompt strategies influence the quality and consistency of IDS rules produced by LLMs?* We explore *zero-shot* prompting vs. *few-shot* and *chain-of-thought* prompts, hypothesizing that more guided prompts may yield more precise but possibly fewer generalizable rules.

RQ3: Appropriate Evaluation Metrics – *What metrics best capture the effectiveness and reliability of LLM-generated IDS rules?* Beyond simple syntax validation, we identify metrics such as the percentage of correctly parsed rules, the incidence of duplicate or redundant rules, and detection performance (*precision*, *recall*, *F1-score* on malicious traffic) to quantitatively evaluate the generated rules' utility.

RQ4: Packet-Level Feature Inclusion – *How does the inclusion of packet-level information —incorporating minimal, intermediate, and comprehensive feature sets— affect the quality and effectiveness of IDS rules generated by LLMs?* Incorporating detailed packet-level features may enhance the precision and contextual relevance of the generated rules. However, it remains unclear how different levels of feature granularity impact rule accuracy and operational robustness.

By answering these RQs, we seek to understand the conditions under which LLMs can serve as effective tools for automated rule generation in industrial cybersecurity and how to best configure prompts and inputs to maximize their performance.

II. BACKGROUND

This section provides context on industrial threats, network analysis, IDS technologies, LLMs in cybersecurity and the simulated environments used in our experiments.

a. Threats in Industrial Environments

Industrial networks (e.g., factory automation, power grids) face unique threats ranging from espionage and sabotage to ransomware. Unlike IT systems, ICS often involve legacy protocols and safety-critical processes, making the impact of cyber attacks potentially severe [2]. Notorious incidents such as the Stuxnet malware targeting uranium enrichment, attacks on Ukraine's power grid, and the Triton malware in petrochemical plants illustrate how adversaries can disrupt or even take control of physical processes. These threats exploit both IT-facing components (e.g., SCADA servers, engineering workstations) and OT devices (PLCs, sensors) to achieve their goals. A major challenge is that many ICS were not originally designed with security in mind, and downtime for patching is limited. This elevates the need for proactive detection of anomalies or known attack signatures before damage is done. Attack vectors include network-borne exploits (malicious packets or commands), unauthorized device access, and lateral movement from corporate networks into OT.

b. Network Traffic Analysis

An essential task in securing ICS is analyzing network traffic for signs of intrusions. Network traffic analysis involves capturing packet data and inspecting it for abnormal patterns or known malicious signatures. In industrial settings, this can be challenging because protocols like Modbus, DNP3, or MMS are less familiar and often proprietary [3]. Tools such as Wireshark provide dissectors for many industrial protocols, allowing security analysts to inspect ICS traffic at the packet level [4]. By examining traffic captures (PCAPs), analysts can identify attack indicators (e.g., a PLC write command outside normal parameters, or malformed protocol sequences). This analysis is foundational for creating IDS rules.

c. Detection Technologies and IDS

Intrusion detection can be broadly categorized into signature-based and anomaly-based methods. Signature-based IDS (like Suricata or Snort) use a database of known attack patterns (rules) to flag matching traffic. They are effective for known threats and yield low false alarms when rules are well-crafted, but they require continuous updates to cover new threats. Suricata is an open-source IDS/IPS that uses rules with a rich syntax to match packet headers, payload content, and flow context [5]. An example rule might look for a specific Modbus function code with a dangerous value in the packet payload. Human experts typically write such rules after analyzing threats. In contrast, anomaly-based IDS utilize machine learning to model normal network behavior and alert on deviations. These can potentially detect novel attacks but often suffer from higher false positive rates and lack clear explanations for alerts. Given the industrial need for deterministic and explainable defenses, signature-based detection remains very relevant. Suricata rules are transparent (explicitly stating what pattern triggers an alert) and can be reviewed by engineers.

d. Simulated Environments

Conducting live cyberattacks on operational industrial systems is infeasible due to safety and availability concerns. Therefore, simulated ICS environments are leveraged to generate realistic attack data in a safe setting. In this work we use the open-source framework ICSSIM [6] to create a virtual testbed that mirrors a plant network. ICSSIM enables researchers to emulate ICS components (PLCs, HMIs, etc.) and their network communications. A specialized simulator for the IEC-61850 protocol (substation automation) called IEC61850openserver [7] was also utilized for generating MMS traffic in one scenario. By using these simulated environments, the background traffic and protocols in the PCAPs are realistic for a factory setting is ensured (including normal operations like sensor readings and control commands).

e. LLMs in Cybersecurity

LLMs have exhibited remarkable ability to generate coherent text and code from prompts. In cybersecurity, researchers have begun exploring LLMs for tasks such as code security auditing, malware description, and even writing simple detection rules [8]. However, applying LLMs to produce correct and safe security configurations poses challenges. One major issue is hallucination, an LLM might output plausible-looking but incorrect or even syntactically invalid content if the prompt is not precise. Ensuring that the model's output conforms strictly to Suricata's rule syntax is non-trivial. Another challenge is the context limitation: LLMs have a token limit, so providing all details of network traffic may exceed what the model can process. Additionally, LLMs do not inherently "know" what makes a good IDS rule –they must infer it from examples or instructions- thus, prompt engineering becomes critical. Another limitations comes from the LLMs' non-deterministic nature (the randomness factor can lead to variations in each run). This required running multiple attempts or using temperature controls to get stable results. Security practitioners are cautious about automated decisions; hence our design keeps a human-in-the-loop for final validation of any LLM-generated rule.

III. RELATED WORK

Automating IDS rule generation has long been pursued to reduce the burden on human analysts. Earlier methods applied data mining to network data to derive rules. For example, *Fallahi et al.* [9] propose an approach using decision-rule algorithms (Ripper, C5.0) to automatically generate Snort rules from flow records. Their system successfully detected certain attacks (DoS, brute force) using flow-level features (IP addresses, ports), but it neglected deep packet payload inspection. This limitation meant complex attacks could evade detection. Our work extends this idea by incorporating packet payload content into the rule-generation process, thereby improving precision against sophisticated threats. *Sagala et al.* [10] uses honeypot logs to produce Snort rules focused on attacker and target IPs and ports. While this static log-to-rule mapping automates response to observed malicious IPs, it lacks context and adaptability, resulting in coarse signatures that miss nuanced attacks. In contrast, our LLM-based approach generates more context-rich rules (including content patterns, protocol fields, etc.) to reduce false negatives and false positives.

More recently, researchers have begun exploring LLMs in cybersecurity. *Louro et al.* [8] studied fine-tuning GPT-style models to generate syntactically correct firewall/IDS rules. They found that general-purpose LLMs can be repurposed for rule generation but often require extensive domain-specific fine-tuning for acceptable performance. Our framework avoids dataset-specific fine-tuning by leveraging powerful LLMs via prompt engineering, aiming for immediate usability with pre-trained models. This design prioritizes real-time applicability (using commercial LLM APIs to draft rules on the fly) and demonstrates that even without fine-tuning, careful prompts can yield deployable rules. *Houssel et al.* [11] explored the use of LLMs to explain IDS alerts, aiming to enhance analyst understanding. They noted challenges such as accuracy and computational demands that limit real-time detection capabilities. This research aligns with their view of LLMs as assistive tools but extends it further by combining rule generation and explainability. Specifically, our framework leverages LLMs not only to create actionable Suricata signatures but also provides clear, concise explanations detailing the rationale and purpose behind each generated rule, enhancing transparency and trust.. Finally, *Tudosi et al.* [12] developed an automated system to dynamically generate firewall rules from IDS logs and alerts. This reduced manual work, but the generated firewall rules were largely broad IP/port blocks, and the system depended on known attack signatures to trigger rule creation. We improve upon this by having LLMs generate detailed and proactive rules triggered by recognizing attack patterns in raw traffic, not just reacting to alerts. In summary, prior studies underscore the potential of automation and LLMs in security, but also reveal limitations; lack of deep packet context, need for fine-tuning, or reactive scope. Our framework builds on these insights, using LLMs to produce more precise, context-aware IDS rules and systematically evaluating their efficacy in an ICS context.

IV. PROPOSED FRAMEWORK

Our framework for automated IDS rule generation consists of a pipeline that processes raw network traffic data, interacts with an LLM to produce candidate rules, and validates those rules before deploying them. The workflow comprises several stages:

Traffic Input & Preprocessing: The process begins by analyzing packet captures from the industrial environment to identify traffic segments associated with suspected attacks. Initially, the Zeek tool extracts flow summaries from the main PCAP file. Subsequently, flows related specifically to anomalous activities are isolated based on attacker IP addresses, effectively filtering them from the broader set of normal flows. These isolated anomalous flows are then structured into concise textual representations, clearly describing the malicious activity in a log-like format. This IP-based labeling is justified by the controlled, single-attacker setup of our simulated testbed, where the source IP unequivocally marks malicious flows. We acknowledge this approach may not generalize to stealthy or multi-host threats (APTs, compromised legitimate endpoints) and thus flag it as a limitation and a direction for future work.

LLM Prompting Strategy: The framework employs a structured, iterative prompting strategy involving multiple calls to the LLM:

1. *First Prompt (Initial Rule Generation):* This initial prompt receives anomalous flow details along with essential parameters (e.g., the last rule SID, LLM model, temperature, prompt strategy —*zero-shot*, *few-sho*t, or *chain-of-thought*). It generates preliminary Suricata rules accompanied by natural-language explanations that clearly articulate the rationale and intended detection logic.

2. *Second Prompt (Evaluation):* The second call re-submits the initial anomalous flows and the generated preliminary rules. Its role is to classify each rule as acceptable, needing further refinement, or discardable. Minor corrections for consistency are applied at this stage, and rules requiring detailed packet-level context for refinement are flagged for the third prompt.

3. *Third Prompt (Refinement):* Rules flagged for refinement undergo detailed packet extraction using Tshark filters based on the rule's attributes, capturing the first 500 bytes of relevant payloads. The rule, its explanation, and the extracted packet details are fed back to the LLM. This enriched context enhances rule precision, aligning detection patterns more closely with actual attack characteristics.

   Note: Since payload data is only used at this stage, rules not selected for refinement by the LLM lack payload context.

4. *Fourth Prompt (Parser-based Correction):* Finally, the refined rules are validated using Suricata's parser in test mode. If syntactic or structural errors arise, the affected rules, along with specific parser error messages, are resubmitted for further correction. This iterative correction cycle continues until all rules pass validation, capped at three iterations.

Rule Parsing and Validation: The text returned by the LLM is parsed to extract actual rule lines. Here the framework ensures that only syntactically valid rules are kept. In the proposed implementation, a regex-based parser was applied to extract the Suricata rule and it's explanation from the LLM's output. The framework automatically removes obvious duplicates in the model's output and ensures each rule is unique. Each candidate rule is then tested with Suricata's parser to check for syntax correctness. This step catches any minor format issues. If the model included non-critical errors, a optionally last call to the LLM could be performed to correct itself as stated in *"Fourth Prompt (Parser-based Correction)*".

Automated Rule Evaluation: Validated rules are rigorously evaluated against network traffic to determine their effectiveness in detecting malicious activities while minimizing false positives. The evaluation methodology integrates several stages, including Ground Truth creation, rule application, and detailed metric analysis at both the packet and flow levels.

Ground Truth Creation: A reliable Ground Truth dataset is established by manually inspecting a full network capture (main PCAP) using Wireshark. At least one representative packet from each suspicious flow is manually labeled and exported, forming a reduced PCAP file that captures confirmed malicious traffic. To guarantee that entire malicious flows are represented, the *Community ID* is leveraged to correlate the exported packets with all associated packets in the original PCAP. Consequently, comprehensive malicious flows are identified and extracted, forming a complete and accurate Ground Truth dataset.

Application of Rules and Alert Generation: Once the Ground Truth is created, the validated Suricata rules generated by the LLM are loaded into a local Suricata instance, which analyzes the complete original PCAP.

Suricata produces alerts based on matches between traffic patterns and rules, logging these alerts into structured JSON files for subsequent analysis. This alert analysis classifies outcomes into four categories: *True Positives* (Malicious flows correctly identified), *False Positives* (Benign flows incorrectly flagged), *False Negatives* (Malicious flows not detected) and *True Negatives* (Benign flows correctly ignored).

Evaluation Metrics: Rule effectiveness is quantitatively assessed using standard performance metrics, including precision, recall, accuracy, and F1-score, computed at both packet and flow levels:

- *Packet-level metrics:* Measure the correctness of each packet classification individually, providing granular insights. However, relying solely on packet-level evaluation can lead to biases, particularly with attacks that involve extensive flows (e.g., DoS or brute-force attacks). Missing detection of a single large flow can disproportionately reduce precision and recall, skewing overall results.
- *Flow-level metrics:* Evaluate whether entire communication flows —comprising multiple packets— are correctly classified. Flow-level evaluation is particularly relevant for industrial environments, where malicious activities often manifest as continuous streams of packets within a single flow.

This combined packet and flow-level assessment ensures that rule evaluations accurately represent real-world attack scenarios and prevents large, undetected flows from significantly biasing the metrics.

V. EXPERIMENTS AND RESULTS

A series of experiments were conducted to evaluate the framework across three representative attack scenarios in a controlled ICS environment. The experiments were three:

1. *First Experiment: Single-Prompt vs. Three-Prompt Strategy:* Compares the effectiveness of a single-prompt versus an iterative three-prompt chain for IDS rule generation. The single-prompt method generates rules exclusively based on flow-level information, whereas the three-prompt strategy introduces iterative refinement with packet-level context, potentially improving accuracy and detection performance.
2. *Second Experiment: Model, Prompt Strategy, and Temperature Impact.* Evaluates how variations in LLM models, prompt strategies (zero-shot, few-shot, chain-of-thought), and temperature settings influence rule generation quality. This experiment identifies optimal combinations of parameters by assessing precision, recall, and the number of necessary rule corrections.
3. *Third Experiment: Packet-Level Feature Inclusion.* Analyzes whether varying levels of packet-level detail (minimal, intermediate, comprehensive) provided to the LLM enhance IDS rule accuracy. The experiment aims to determine if richer contextual inputs significantly improve rule effectiveness or introduce unnecessary complexity.

Each scenario represents a distinct attack type(s) recreated in the simulated ICS testbed. Within each scenario, every possible experimental configuration —defined by a distinct set of parameters such as *prompt strategy*, *model choice*, *temperature setting*, and *packet-detail granularity*— is evaluated, with 11 iterations performed for each configuration. For each attack scenario, the corresponding PCAP is collected, all configurations are tested, and the generated rules are then assessed for their ability to detect the threat in that same scenario. Presenting results by scenario highlights how rule performance varies both with attack type and with configuration choice.

Scenario 1: SQL Injection + Modbus PLC Write.

This scenario mimics an attacker who first exploits a vulnerable web interface (SQL Injection) to gain a foothold and then issues unauthorized Modbus/TCP commands to a PLC. The combined attack was executed in the ICSSIM environment.

For the *GPT models*, under the one-prompt strategy —where only flow-level details are provided— GPT-4o-mini and o3-mini achieve 100% parsing accuracy *(without the use of the Suricata's parser feedback and last LLM query for correction)*, while GPT-4o reaches approximately 75%. With the three-prompt chain, which incorporates packet-level details for increased rule precision, GPT-4o-mini's parsing accuracy decreases to around 75%, and both GPT-4o and o3-mini drop to 0%. *Only the* GPT-4o-mini *generates duplicate rules*. When the whole pipeline is executed (the three prompts and the parser's feedback) the *zero-shot* strategy achieved the highest *recall* (~64.7%) and strong overall *F1 scores*, indicating broader detection capabilities but at the cost of slightly reduced syntax *accuracy* (~97.7% rules parsed correctly). Conversely, the few-shot strategy produced the most syntactically accurate rules (~99.7% parse success) but significantly limited detection (recall ~32.4%), creating overly specific rules based on provided examples. The *chain-of-thought* strategy balanced these extremes, offering moderate recall (~51.6%) and high syntax correctness (~97.0% parsed), providing stable and iterative refinement. Incorporating intermediate packet-level features boosts accuracy and recall while avoiding the overfitting seen with all features; minimal features result in overly broad rules that miss attacks.

When using a single prompt, *gemini-2.0-flash* achieves about 76% parsing accuracy, *gemini-2.0-flash-lite* around 65%, and *gemini-1.5-pro* 100%. Shifting to the three-prompt approach, *gemini-2.0-flash drops* to roughly 35%, while *gemini-2.0-flash-lite* and *gemini-1.5-pro* each settle near 50%. Regarding duplicates, the 9% of the generated rules by *Gemini 1.5-pro* model were duplicates, reaching percentages of 28% in some iterations. When executing the whole pipeline, the *chain-of-thought* strategy was optimal for Gemini, achieving the highest recall (~68.7%), outperforming zero-shot (~49.2%) and few-shot (~27.7%) significantly. However, Gemini consistently encountered more parsing errors (~87.6% parsed), indicating lower reliability in syntax compliance. Gemini also generated more duplicates and higher variance in rules created, highlighting greater unpredictability compared to GPT and Claude.

In the case of Claude's models, *Claude-3.5-Haiku* achieves about 50% parsing accuracy, while *Claude-3.7-Sonnet* attains around 65%. Under three prompts, *Claude-3.5-Haiku* drops to 0%, and *Claude-3.7-Sonnet* ranges between 55% and 60%. Neither Claude model produces any duplicate. When executing the whole pipeline, the **few-shot** strategy greatly improved recall (~61.2%) compared to zero-shot (~26.9%), though at the expense of parsing accuracy (~94.6%). Claude performed best syntactically in zero-shot (~98.9% parsed) but detected few attacks. Chain-of-thought prompting provided a good middle ground (recall ~53.3%, parsing ~97.9%), suggesting Claude benefits significantly from additional reasoning or examples to enhance detection without major parsing issues. Performance improve with richer packet-level details; while intermediate features offer a solid balance, including all available details further enhances recall and F1-score compared to minimal configurations.

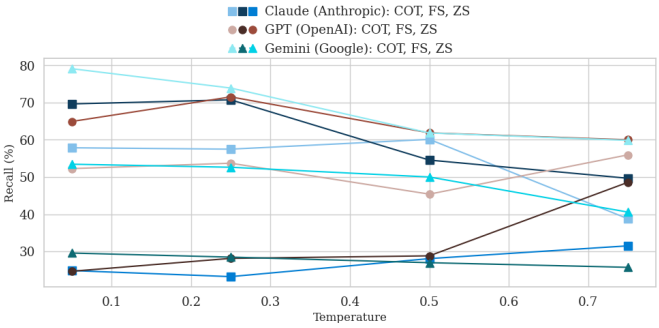

Figure 1: Recall versus Temperature by Family Models for the Scenario 1

Table 1: Average Flow Metrics by Prompt Strategy (aggregated across all models)

Prompt Strategy	*Precision (avg.)*	*Recall (avg.)*	*F1-score*
Zero-shot	~0.89	~0.41	~0.50
Few-shot	~0.94	~0.34	~0.44
CoT	~0.99	~0.44	~0.55

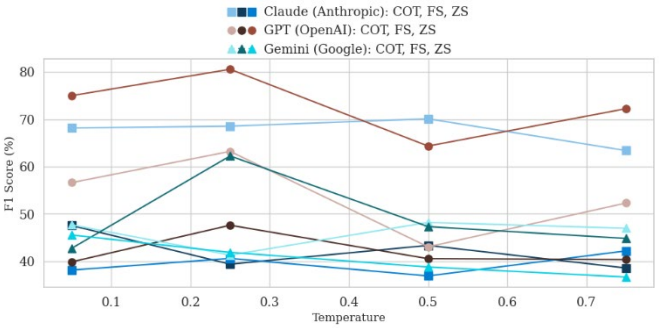

Figure 2: F1-Score versus Temperature for each Model Family and Prompt Strategy (NMAP Scan)

Table 2: Average percentage of rules parsed correctly and Average number of duplicate rules

Model Family	Single-Prompt **Parsed %**	Three-Prompt **Parsed %**	Single-Prompt **Duplicates (avg)**	Three-Prompt **Duplicates (avg)**
GPT	90.9% (±15.3)	65.8% (±37.5)	3.4 (max 102)	0.9 (max 47)
Claude	58.3% (±5.9)	23.4% (±30.2)	0.0	0.0
Gemini	82.2% (±12.2)	41.9% (±34.7)	0.0	0.2 (max 4)

Scenario 2: Nmap Scan of ICS Network.

The third scenario is an active reconnaissance attack: an Nmap port scan sweeps the industrial network to map out open services on PLCs and HMIs. This is a common precursor to targeted attacks. Detecting scans is a classic IDS task; however, this scenario was used to evaluate how LLM-generated rules handle high-frequency events and how they utilize different levels of detail. The network capture contained numerous TCP SYN packets and ARP requests from the scanner.

For the *GPT models*, the non-reasoning variant, *gpt-4o-mini*, proved to be quite conservative with an average *recall* around 28% and an *F1-score* near 0.42, even though in lower temperature *zero-shot* settings it reached roughly 50% *recall* (*F1* ~0.65). In contrast, the reasoning-enhanced *o3-mini* model outperformed its counterpart, achieving about 65% recall and

an F1-score of approximately 0.71 across different temperature settings, while maintaining nearly perfect precision. *Zero-shot* prompting produced concise and consistent rule sets with minimal duplicates, whereas *few-shot* prompting resulted in larger, more variable outputs. The *chain-of-thought* strategy generally led to the highest detection performance —with average *recall* around 44% and an *F1-score* near 0.55— by allowing the model to reason through complex scanning patterns. Additionally, experiments on packet-level feature inclusion indicated that providing *minimal* details (focusing on core features like source/destination IPs, ports, and SYN packet indicators) yielded the best results for GPT models; adding more packet-level information slightly reduced recall (from ~61.4% to ~54.7%) without significant gains in precision.

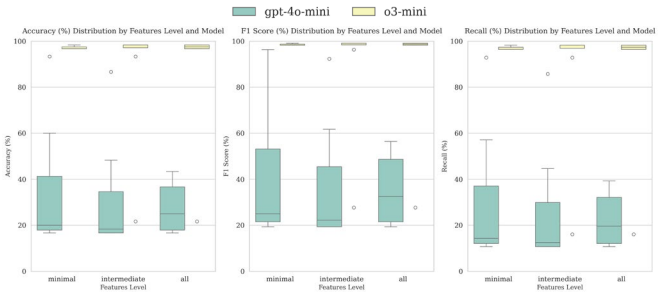

Figure 3: Accuracy, F1-Score and Recall Boxplot for GPT Models for the Scenario 2

Within the *Gemini* family, performance was more variable. The *Gemini-1.5-Pro* model achieved roughly 49% *recall* and an *F1-score* around 0.59 with excellent *precision* (~97.5%), outperforming the newer *Gemini-2.0-Flash* variant, which reached about 40.7% recall and an *F1-score* of 0.49, with *precision* averaging ~85%. The lightweight *Gemini-2.0-Flash-Lite* model struggled considerably, managing only around 15% recall and an F1-score near 0.21, with lower *precision* (~72%). In some cases, *few-shot* prompting marginally improved *recall* (up to ~39.6%) for certain Gemini variants, while *chain-of-thought* prompting boosted detection in low-temperature runs from near-null values to 0.3–0.4; however, these strategies also led to increased duplicate rules and greater variability in outputs.

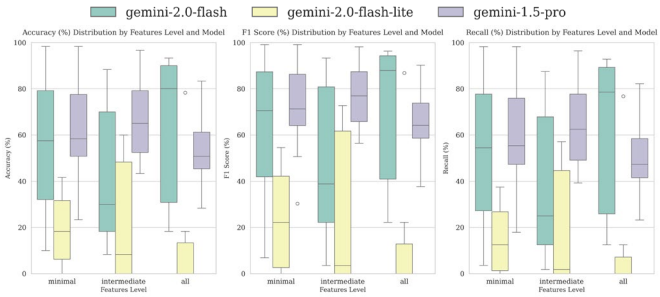

Figure 4: Accuracy, F1-Score and Recall Boxplot for Gemini Models for the Scenario 2

For the Claude models, the advanced *Claude 3.7 Sonnet* clearly outperformed the lighter *Claude 3.5 Haiku* variant, achieving about 47.5% *recall* and an *F1-score* around 0.60 compared to ~26% *recall* and an *F1* of 0.40 for the latter. Both Claude models maintained near-perfect *precision* (virtually 100%) across runs. Under *chain-of-thought* prompting, *recall* for *Claude 3.7 Sonnet* improved dramatically in some cases (up to ~85%), though overall, *few-shot* prompting tended to yield the lowest *recall* (~34%) despite a slight precision advantage. Consistently across the experiments, Claude models benefited most from *minimal* packet-level input; adding excessive packet details introduced noise that significantly reduced both *precision* and *recall*.

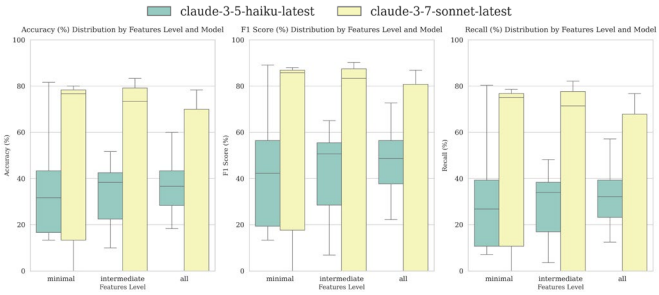

Figure 5: Accuracy, F1-Score and Recall Boxplot for Claude Models for the Scenario 2

VI. DISCUSSION

Our findings demonstrate the viability of LLMs for automating IDS rule creation in industrial contexts, while also revealing important considerations. To revisit the research questions: *RQ1 (Effectiveness)* – The experiments confirmed that state-of-the-art LLMs can generate syntactically correct and operationally effective Suricata rules for diverse ICS threats. In each scenario, at least one LLM produced a high-quality rule or rule set that successfully detected the attack without expert intervention. Notably, certain models stood out: *Claude 3.7* (Anthropic) and *GPT o3-mini* (OpenAI) consistently yielded accurate rules when given clear prompts, detecting even sophisticated multi-step attacks. We observed that Claude models tended to be more deterministic and produced fewer errors, which suggests their training favored reliability – an attractive trait for security applications. Gemini (Google) models generated valuable rules as well, though with higher variability; they sometimes offered very concise solutions and other times overly verbose ones, indicating that iterative prompting or temperature tuning was needed. In summary, *RQ1* is answered affirmatively: LLMs (especially Claude and GPT) can indeed serve as an effective "*virtual analyst*" to write IDS rules, given proper configuration.

For *RQ2 (Prompt Engineering)*, our results highlight that prompt strategy significantly influences outcomes. A *zero-shot* approach was simpler and often yielded a correct basic rule, showing higher initial parse success rates across all model families. However, these rules were sometimes over-generic (catching the obvious pattern but potentially missing subtle variations, or creating redundant rules). Using a multi-step chain-of-thought prompt forced the model to be more thorough and reduced redundant outputs (we saw dramatically fewer duplicate rules with chain-of-thought). The trade-off was that the added complexity could confuse the model's rule formatting, lowering the immediate parsing success. Few-shot prompting (providing examples) proved beneficial for syntax. Models almost never made format mistakes after seeing an example, and correctness was high. However, those examples can bias the model towards copying the specifics of the example, sometimes making rules too narrowly tailored

("overfitting" to the example's pattern). The key insight is that there is no one-size-fits-all prompt: one must balance simplicity for correctness and complexity for completeness. In practice, a hybrid approach may work best: e.g., first use a zero-shot prompt to get a base rule, then use a refined prompt to improve it. Security analysts can then choose which result to deploy. This finding underscores prompt engineering as a crucial skill for applying LLMs in cybersecurity tasks.

Addressing *RQ3 (Evaluation Metrics)*, we found a combination of syntactic and semantic metrics necessary to fully assess LLM-generated rules. Syntactic parsing accuracy (percentage of rules that load without error) was a fundamental metric; in our tests, parsing accuracy above ~90% was needed for a model to be practically useful. All top-tier models met this threshold under optimal prompts, with Claude reaching ~100% in many cases. We also measured the number of duplicate rules and any Suricata parser errors as indicators of output quality. High duplicate counts indicate the model might be "spamming" patterns, which could burden an IDS with redundant checks. Our multi-scenario tests showed that after prompt tuning, duplicates were minimal for most models (often zero or one duplicate at most). Importantly, we introduced flow-level detection metrics (*precision*, *recall*, *F1-score* on malicious vs. benign flows) to directly evaluate if the rules perform their security function. This bridged a gap: a rule that parses correctly isn't useful unless it actually catches attacks and ignores normal traffic. By combining these metrics, we could identify the best configurations. Overall, the chosen metrics effectively captured both the technical correctness and the operational efficacy of generated rules, they can serve as a template for future evaluations of AI-generated security rules.

For *RQ4 (Packet-Level Feature Inclusion)*, the experiments revealed a nuanced but clear pattern: intermediate detail is optimal. When LLMs were given only minimal info, the resulting rules lacked precision or missed specifics (low recall/F1). Conversely, when flooded with all packet bytes or exhaustive details, models occasionally fixated on irrelevant aspects or became inconsistent in formatting, hurting parsing accuracy and sometimes recall. The intermediate approach – providing the most relevant fields and a bit of context– produced the best outcomes. This likely works because it gives the LLM enough context to understand the attack (nuanced behaviors) but not so much that it loses focus or runs out of its attention span. Practically, this means that when using LLMs for rule generation, one should carefully select what input data to include.

Beyond the *RQs*, we observed some model-specific trends worth noting. GPT models emerged as strong overall performers, they often produced effective rules across all scenarios, showing versatility. They did, however, sometimes require the chain-of-thought prompt to reach that effectiveness, indicating they respond well to reasoning tasks. Claude models were remarkably good at maintaining Suricata syntax and providing what felt like professionally written rules. Their deterministic nature (less randomness) is a double-edged sword: great for consistency, but if they misunderstood the prompt, they would consistently do so. We mitigated that by prompt clarity. Gemini models improved significantly with iterative prompting; their first attempt might be messy, but they quickly adapted, which suggests that an interactive loop proposed can harness them effectively. In terms of speed and cost, smaller models (like *GPT-4o-mini* or *Claude 3.5 Haiku*) were faster and cheaper to run, yet still yielded decent rules. This raises an operational point: one could use a cheaper model for quick initial rule drafts and then validate or refine with a more powerful model if needed.

From a security operations perspective, these results indicate that an LLM-driven rule assistant could soon become a reality. An analyst could feed network logs of a new attack into such a system and get a candidate rule, saving precious time during incident response. Our framework's output is human-readable rules, meaning the final decision remains with analysts. They can inspect the AI-generated rule and decide to deploy it, combining AI speed with human judgment. We also note that instructing LLMs to consider negative cases (what not to alert on) by embedding domain knowledge, is useful.

Finally, we must consider limitations. One limitation is token/context size (complex scenarios might exceed the LLM's input limit). We managed this by trimming inputs, but as attacks get larger, this could be problematic (discussed in section VII). Another limitation is that our evaluation was on a controlled set of attacks; real-world traffic might introduce noise that could confuse the models or cause false positives. We did not deeply test the robustness of rules against slight attack variations (beyond the ones the LLM itself considered).

However, given the generally specific nature of the rules, they might need updates if attackers significantly change tactics. Encouragingly, the LLM can simply be re-prompted with new data in such cases. The framework is not yet fully autonomous, we foresee it as a recommendation system for human analysts rather than a black-box rule generator deployed without oversight.

VII. FUTURE LINES

While the framework is effective, there are several avenues to explore to further mature this approach for real-world deployment:

Handling Large or Continuous Data: One immediate area for improvement is dealing with LLM token limitations. Future research can investigate intelligent data segmentation (breaking traffic into chunks that the LLM can process sequentially, perhaps with a rolling context). Techniques like summarizing or clustering similar traffic can reduce input size without losing important information. As newer LLMs with larger context windowsbecome available, the framework should be updated to leverage those, feeding them more complete views of traffic. Maintaining a modular design will allow incorporating such advanced models seamlessly. The ultimate goal is to handle live data streams, potentially by sliding window analysis where the LLM processes recent traffic slices for rule generation suggestions.

Real-time and Autonomous Operation: Currently, a human validates and deploys the LLM-generated rules. Moving towards real-time applicability, we envision integrating this framework with a digital twin or testbed environment for automatic validation. For example, before pushing a rule to a production IDS, it could be tested in a virtual replica of the network (using a simulation) to ensure it catches the intended

malicious behavior and does not over-trigger. This automated testing loop could enable a more autonomous system that generates and verifies rules on the fly. Additionally, research into reducing the need for manual oversight through more robust self-checks is warranted.

Improving Consistency (Reducing Non-determinism): The stochastic nature of LLM outputs means each run might yield a different rule. While we found ways to minimize this (low temperatures, etc.), completely eliminating variability would boost trust. Future work could explore fine-tuning an LLM on cybersecurity data –e.g., training it with lots of example attacks and correct rules– to see if that yields more consistent and domain-optimized outputs. Fine-tuning might also reduce prompt complexity needed, as the model internalizes how to map traffic to rules. Techniques like programmatic chain-of-thought, where the model's reasoning is constrained, could yield repeatable results. We believe achieving higher predictability is crucial for critical infrastructure security adoption, and it's an active area for model improvement.

Robust Output Parsing and Format Handling: We relied on regex parsing to extract rules from the LLM output, which worked but required maintenance (e.g., updating the regex when a model formatted output unexpectedly). Future iterations could employ more sophisticated techniques like using an LLM or parser to interpret the output. Alternatively, incorporating a suricata rule grammar into a parser or using ML-based parsing could make the extraction step more robust to variations. By improving this aspect, the system becomes less brittle and can handle any formatting quirks from future models.

Expanding to Other Security Domains: While our focus was Suricata IDS rules, the concept can extend to other rule-based systems – for example, firewall rules, SIEM alert rules, or even detection code (like Snort dynamic preprocessor or YARA rules for malware). Future work could test LLMs on generating those artifacts, potentially using a similar approach. Each domain has its syntax and semantics, so some adaptation in prompt and validation would be needed.

On-premise and Privacy-Preserving Models: One practical direction is deploying this capability in environments where sending data to cloud APIs is not acceptable (common in critical infrastructure due to privacy). Future lines include evaluating the performance of open-source LLMs that can run on local hardware. If a smaller fine-tuned model can run on an edge server and produce rules nearly as well as those generated by the models tested, that would be a significant win for adoption. Techniques like model distillation or quantization might be applied to compress the knowledge of a large model into a lightweight one that an organization can use internally.

The next steps aim to make the LLM-driven IDS rule generation more scalable, autonomous, and integrated. By overcoming current limitations –token limits, need for human review, output variability– we move closer to an era of self-updating IDS, where new threats are rapidly countered by AI-crafted defenses. Our work provides a foundation, and these future directions outline a path to transition this prototype into a robust, field-ready system. We anticipate that as LLM technology and cybersecurity datasets evolve, the synergy between them will lead to increasingly intelligent and reliable security infrastructure, reducing the window of exposure to new threats and easing the burden on human analysts.

ACKNOWLEDGMENTS

This work has been supported by CRITIC Project Grant PLEC2024-011222 funded by AEI/10.13039/501100011033, FEDER and EU.

BIBLIOGRAPHY

[1] Kaspersky, «Threat landscape for industrial automation systems. Statistics for H1 2021,» 9 September 2021. [Online]. Available: https://ics-cert.kaspersky.com/publications/reports/2021/09/09/threat-landscape-for-industrial-automation-systems-statistics-for-h1-2021/.

[2] R. Masum, «Cyber Security in Smart Manufacturing (Threats, Landscapes Challenges),» *arXiv,* 20 April 2023.

[3] J. F. &. R. K. W. Kurose, «Computer Networking: A Top-Down Approach (8th ed.),» Pearson, 2021.

[4] Wireshark, «Wireshark Documentation,» [Online]. Available: https://www.wireshark.org/docs/.

[5] A. Y. H. L. Y. S. a. L. S. Yan Hu, «A survey of intrusion detection on industrial control systems,» *SAGE,* p. 14, 2018.

[6] A. Dehlaghi, «ICSSIM | GitHub,» [Online]. Available: https://github.com/AlirezaDehlaghi/ICSSIM.

[7] robidev, «iec61850_open_server | GitHub,» [Online]. Available: https://github.com/robidev/iec61850_open_server.

[8] R. A. J. C. C. J. B. F. S. a. P. R. M. I. Bernardo Louro, «Analysis of the Capability and Training of Chat Bots in the Generation of Rules for Firewall or Intrusion Detection Systems,» *International Conference on Availability, Reliability and Security,* p. 7, 2024.

[9] A. S. a. M. Naser Fallahi, «Automated Flow-based Rule Generation for Network Intrusion Detection Systems,» *Iranian Conference on Electrical Engineering (ICEE),* 2016.

[10] A. Sagala, «Automatic SNORT IDS Rule Generation Based on Honeypot Log,» *International Conference on Information Technology and Electrical Engineering (ICITEE),* 2015.

[11] P. S. S. L. a. M. P. Paul R. B. Houssel, «Towards Explainable Network Intrusion Detection using Large Language Models,» *arXiv,* 2024.

[12] A. G. D. G. B. A. D. P. a. R.-C. T. Andrei-Daniel TUDOSI, «Design and Implementation of an Automated Dynamic Rule System for Distributed Firewalls,» *Advances in Electrical and Computer Engineering,* p. 10, 2023.