Proceedings of Machine Learning Research 299:1–16, 2025 Conference on Applied Machine Learning for Information Security

Evaluating LLM Generated Detection Rules in Cybersecurity

Anna Bertiger ANNA.B@SUBLIMESECURITY.COM
Bobby Filar BOBBY@SUBLIMESECURITY.COM
Aryan Luthra ARYAN@SUBLIMESECURITY.COM
Stefano Meschiari STEFANO@SUBLIMESECURITY.COM
Aiden Mitchell AIDEN@SUBLIMESECURITY.COM
Sam Scholten SAM@SUBLIMESECURITY.COM
Vivek Sharath VIVEK@SUBLIMESECURITY.COM
Washington, DC, USA

Editor: Edward Raff and Ethan M. Rudd

Abstract

LLMs are increasingly pervasive in the security environment, with limited measures of their effectiveness, which limits trust and usefulness to security practitioners. Here, we present an open-source evaluation framework and benchmark metrics for evaluating LLM-generated cybersecurity rules.

The benchmark employs a holdout set-based methodology to measure the effectiveness of LLM-generated security rules in comparison to a human-generated corpus of rules. It provides three key metrics inspired by the way experts evaluate security rules, offering a realistic, multifaceted evaluation of the effectiveness of an LLM-based security rule generator.

This methodology is illustrated using rules from Sublime Security's detection team and those written by Sublime Security's Automated Detection Engineer (ADÉ), with a thorough analysis of ADÉ's skills presented in the results section.

Keywords: evaluating LLMs, agents, security rules

1. Introduction

AI agents are popular, and vendors are latching on the to the idea that idea that they will revolutionize the detection experience. But agents are not without their skeptics, with experienced detection engineers concerned that they produce "plausible sounding nonsense". Like hiring a new employee, it's important to measure whether the agent can do the job. Without these measurements, we can't build trust in the agents' work, and thus will not save any time or energy by using them.

This paper describes a methodology for measuring the effectiveness of LLM generated security rules, and illustrates this method using Sublime Security's Automated Detection Engineer, ADÉ, which is an agentic system that writes queries in Message Query Language (MQL) to detect malicious emails. We evaluate the quality of queries by measuring detection accuracy of samples flagged by the query using a database of labeled email samples, and measuring the robustness of queries written. In addition, we measure the cost of operating the system to optimize cost vs. benefit.

We also compare the quality of queries written by ADÉ with existing human-developed detection rules that experts at Sublime have published and maintained to help protect users

(Sublime Core Feed). We choose a selection of rules, and, one at a time, remove a rule from ADÉ's knowledge base, and ask ADÉ to construct a detection for a true positive (TP) based on that rule.

2. Background and Related Work

Measuring the quality of LLM output is notoriously difficult and context dependent; see, for example, the survey of Chang et al. (2024), and the recent paper on measuring LLM based systems from Rudd, Andrews and Tully Rudd et al. (2025). Measuring the quality of automatically generated security detection rules is no exception.

We draw inspiration from two main sources: the literature that measures the quality of LLM-generated code and the literature that measures the quality of cybersecurity detections. In addition, we take some inspiration from supervised machine learning, by holding out human-created rules from ADÉ's knowledge base in order to compare ADÉ to a human expert. Unlike with supervised learning, this is not entirely a fair comparison, as the human-generated rules are designed by humans fitting a broader collection of data and total background domain knowledge than ADÉ is given.

The code quality literature has a number of papers focused on creating robust test sets where code quality is measured by the ability to change existing tests from failing to passing starting with existing code, test cases and issue descriptions, for example (Jimenez et al., 2024) and (Zhang et al., 2025b). These measurements produce leader boards where LLM-based coding agents compete to produce the best code. Chen et al. (2024) examine LLM performance in repository-level program repair tasks, revealing significant gaps between isolated code generation and real-world debugging scenarios that require understanding of a broader codebase.

Recently, we have also seen the development of code robustness measurements. Li et al. Li et al. (2025) develop a measurement of the robustness of LLM-generated code based on counting conditional statements, which are assumed to increase robustness by checking for edge cases and try/except statements, which are assumed to indicate a lack of robustness if they are needed.

2.1. Evaluating LLMs in Code Generation Tasks

Research into quantifying the output of code-generation LLMs has grown rapidly from measuring the similarity of generated code to that of an oracle to capture functional correctness. The work of Chen et al. (2021) introduced the HumanEval benchmark, along with the *pass@k* metric, which measures the probability that one of the K-generated samples passes a series of unit tests.

The concept of functional correctness has become foundational in the evaluation of code generation models. However, researchers continue to expand this metric to better reflect the real world concerns of introducing LLM-generated code into production environments (Jimenez et al., 2024)(Zhang et al., 2025b).

2.1.1. Secure Code Generation

Currently, there is a growing body of work that measures the security of LLM-generated code. Benchmarks such as CWEval(Peng et al., 2025) and CodeGuard+(Fu et al., 2024) highlight the importance of capturing both functional correctness and security of code generation tasks. This is accomplished using an expanded $pass@k$ metric that penalizes solutions that are secure but functionally incorrect (e.g., a no-op solution). Most recently, Dilgren et al. (2025) presented SecRepoBench, which evaluates LLMs on secure code generation within real-world repository contexts, highlighting the importance of testing generated code in production-like environments.

2.1.2. Code Robustness

Moving beyond the generation of secure code, research such as the RobGen(Li et al., 2025) framework introduces metrics and techniques to evaluate the generated code's ability to handle edge cases or incorrect input, capturing the code's robustness to unexpected scenarios. Likewise, earlier work by Wang et al. (2022) introduced ReCode, which evaluates the robustness of code generation models through perturbation-based testing, establishing foundational methods for assessing the reliability of generated code.

Similarly, Pasini et al. (2024) demonstrate that LLM-generated code often exhibits brittleness that can be evaluated and improved through iterative refinement, strengthening our approach to measuring the robustness of detection rules.

2.2. LLMs for Cybersecurity Tasks

The use of LLMs in the cybersecurity space has also experienced rapid growth, with applications in both offensive and defensive tasks.

For offensive applications, Dawson et al. (2025) introduce AIRTBench, which measures autonomous AI red teaming capabilities in language models, evaluating their ability to identify and exploit vulnerabilities. This work highlights the dual-use nature of LLMs in security, where the same capabilities that enable the generation of defensive rules can also be used for adversarial testing. Fang et al. (2024) demonstrate that LLM agents can autonomously hack websites, discovering and exploiting vulnerabilities without human intervention. These offensive capabilities underscore the arms race nature of cybersecurity, where defenders must anticipate LLM-powered attacks when designing detection rules.

On the defensive side, key applications include threat intelligence, vulnerability and malware analysis, phishing detection, and incident response. These applications focus on applying LLMs to distill unstructured text, classification, and agentic automation tasks. Fu et al. (2025) recently introduced RAS-Eval, a comprehensive benchmark to evaluate security-focused LLM agents in real-world environments, which aligns with our goal of realistic evaluation, but focuses on broader security tasks beyond rule generation.

In addition to code quality metrics and datasets, Kapoor et al. (2024) take a cost benefit analysis point of view, that takes into account not only the benefits in increasing ability to fix bugs, but also the cost of such benefits in dollars to run the LLM.

As with tests passing or failing for code quality, the existing metrics for intrusion detection systems have largely revolved around accuracy in terms of detection rates, true and false positive rates, and false negative rates (which, of course, are very hard to estimate);

see, for example, (Milenkoski et al., 2015). In real life, false negative rates are very rarely counted, as they are incredibly hard to track; after all, we do not know what we do not know. In addition, in practice, true positives that do not appear as results for any other detection rule are often valued over rules with similar precision and recall that do not have this property.

The work builds on these foundational works, while making several novel contributions by introducing a benchmark framework specifically to address the challenges posed by autonomous detection rule generation in the security space. These contributions include:

- **Measuring the Operational Viability of Detection Rules:** Previous work is mainly focused on functional correctness and the generation of secure code. Our framework introduces an end-to-end pipeline to evaluate detection rules holistically. We combine detector validation (e.g. functional correctness) with a real-world corpus for retro-hunting to surface operational noise thresholds, expected maintainability, and, ultimately, performance guardrails, all of which are integrated into a final evaluation score.

- **Built-in Adversarial Robustness:** To better address the brittleness commonly associated with security detection rules, our benchmark incorporates logic to measure brittleness and provide feedback to the code generation agent in an effort to reduce trivial adversarial bypasses.

- **Open-Source Evaluation Framework:** Finally, we plan to open-source the complete benchmark in the hope that it provides a framework for security researchers to test their own rule generation agents (e.g., custom domain-specified language, YARA, etc.). We have provided an adaptable modular pipeline to promote reproducible research and a standardized set of metrics to evaluate code generation agents in the security space.

3. Autonomous Detection Engineer, ADÉ

ADÉ (pronounced "Ah-Day") is Sublime Security's LLM-based system to autonomously generate email threat detection rules in MQL. This agentic system is given access to the same tools and much of the same knowledge available to human detection engineers at Sublime Security in order to function as an autonomous detection engineer. ADÉ produces rules to be approved by a reviewer in much the same way that a human would ask a reviewer to review a PR. Unlike a human engineer, ADÉ is restricted to investigating a particular email and creating or modifying a rule inspired by that message.

ADÉ employs a model-agnostic architecture built around a curated knowledge base containing detection engineering best practices, existing rule patterns, known attacker behaviors and TTPs, and comprehensive MQL documentation. Beyond the knowledge base, ADÉ has access to validation tools, including dynamic file analysis, computer vision, natural language understanding (NLU) capabilities, sender reputation analysis, and comprehensive threat hunting tools. ADÉ can delegate specialized subtasks to focused subagents for activities such as rule critique and hunt result analysis, creating a sophisticated multi-agent workflow. This "Knowledge, Tools, and Subagents" architecture reflects a strategic decision

to avoid traditional fine-tuning approaches that bind system capabilities to specific model weights. Instead, this design choice allows ADÉ to be used with any foundation model.

When ADÉ receives a malicious email sample, it follows a systematic investigative process. The system first conducts a comprehensive analysis of the email, examining headers, content, attachments, links, and sender characteristics to identify specific attack vectors and indicators of compromise. It leverages Sublime's analysis tools, including dynamic file analysis for attachments, link analysis for URL examination, and sender profiling to understand the email's context within the recipient's environment. Based on this investigation, ADÉ identifies key TTPs and indicators that could serve as detection points, such as suspicious sender patterns, content obfuscation techniques, or malicious attachment characteristics.

Following this analysis, ADÉ searches Sublime's existing rules for detection rules targeting similar threats or employing comparable indicators. This research phase allows the system to understand existing detection approaches and identify potential coverage gaps. ADÉ then generates candidate MQL detection rules, drawing from its knowledge base of detection patterns while incorporating the specific indicators discovered during its investigation. The system creates rules that target multiple attack vectors simultaneously, such as combining sender reputation checks with content analysis and attachment scanning to maximize detection effectiveness.

The key to ADÉ's approach lies in its iterative refinement capability to address a fundamental challenge in long-context agentic tasks: maintaining focus and achieving convergence without losing track of objectives. Traditional LLM agents struggle with extended workflows because they lack mechanisms to receive corrective feedback when intermediate steps fail (Cemri et al., 2025). ADÉ solves this through continuous integration of evaluation framework components as feedback signals. After generating initial detection rules, ADÉ validates syntactical correctness through automated MQL syntax checking, tests rules against the original malicious sample, and initiates threat hunts across historical email data. Different subagents then analyze hunt results to provide detailed feedback on detection accuracy (true positive rates), operational noise (false positive rates), and rule robustness. This creates a closed feedback loop where error messages and evaluation metrics become guidance signals, allowing ADÉ to iteratively refine detection logic, adjust false positive reduction conditions, and incorporate additional indicators until converging on effective rules with acceptable hunt results. This feedback-driven convergence mechanism enables ADÉ to complete complex, multi-step detection engineering tasks that would otherwise fail due to context drift or objective misalignment.

Measuring ADÉ's rule quality is critical for ADÉ's continued development. Without clear metrics defining what constitutes a "good" detection rule, and how ADÉ's rules compare to human written rules, it becomes impossible to systematically improve ADÉ's rule generation or validate its effectiveness against human-generated baselines. Indeed, explicitly telling ADÉ what a "good" rule looks like is critical to the system; for example, ADÉ has a tool to hunt and analyze the results exactly to work on the true positive vs. false positive quality of its results.

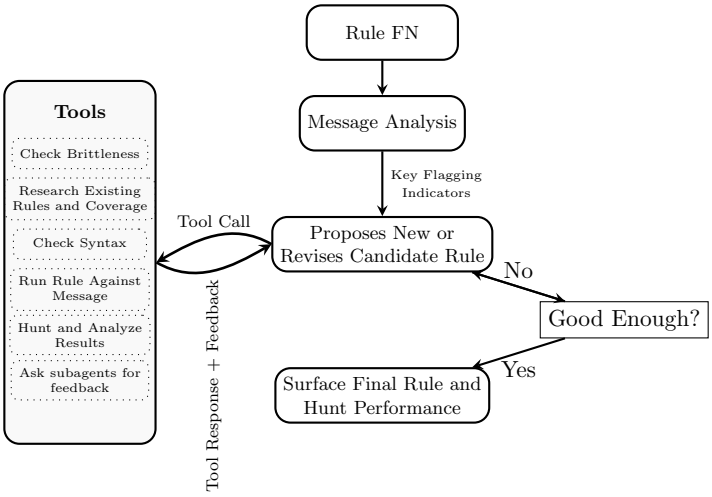

Figure 1: ADÉ (Autonomous Detection Engineer) Workflow

4. LLM Detection Measurement Framework

The main goal of this work is to present a framework for measuring the quality of a detection generated by an LLM in comparison to held out detection rules generated by humans. We measure the quality of a detection by measuring the ability of the detection to find malicious emails, the robustness of the query, and the economic efficiency of the query separately. We then use "holdouts" of human written detections to compare the quality of ADÉ written detections with a known, high quality baseline.

It should be noted that the rules written by experts at Sublime Security are meant to flag email messages that could be malicious, but are not typically a trigger for an action based on the rule alone. They instead trigger further triage, by either a model or a human analyst, so while precision is important, it is not the only metric on which we base our measurements, and we do not expect rules to be perfect in precision, even though we strive to reduce false positves.

In this section we give detailed descriptions of the accuracy metrics that we use to judge rules and intuition about why we chose these metrics. In the results section, we present the results of these metrics on rules held out from ADÉ.

4.1. Detection Accuracy

The foremost concern of any system of security rules, no matter whether generated by humans of AI, is detection accuracy. We measure detection accuracy by measuring the number of true and false positives a rule generates. We are able to measure these counts using an expert human-labeled dataset of emails maintained by Sublime Security for research and development purposes. Once we have ascertained what messages a rule flags and whether those messages are true positives (malicious) or false positives (benign), we also check whether true positives are unique to the detection rule at hand, that is, whether they have been found by any other rules in the core feed, with the idea that malicious emails that are not found any other way are of additional value over ones that are flagged by many

rules. In the end, we compute a score of the effectiveness of a rule by

$$\text{Score} = \frac{1}{2}\left(\frac{\#\text{TP}}{\#\text{TP} + \#\text{FP}} + \frac{\#\text{unique TP}}{\#\text{TP} + \#\text{FP}}\right) \tag{1}$$

This is the average of the standard precision score and the precision of unique true positives, not found by any other rule. It is an attempt to balance the rule's ability to find unique true positives, and thus increase the system's overall recall with the standard classifier effectiveness measurement of precision. In security, we often struggle with our ability to measure false negative rates, as we don't know what we don't know. It is especially hard to understand false negatives in ADÉ's case, as we have not given criteria as to what the rules generated by ADÉ "should" catch. Thus, we focus on the more typical, precision-based detection metrics.

4.2. Economic Cost of Syntactic Correctness

As mentioned in section 3 above, the pipeline includes an MQL validation step that must be passed before the agent can proceed to the retro-hunt evaluation. Failing to pass this validation step causes the agent to enter a potentially costly retry loop, which impacts both computational and economic resources. The economic implications of LLM-based coding are further explored by Miserendino et al. Miserendino et al. (2025), who investigate whether frontier LLMs can perform cost-effective "freelance" software engineering tasks, providing additional context for our cost-benefit analysis.

The *pass@k* metric(Chen et al., 2021) has become a standard to measure *functional* correctness for code generated by an LLM agent. However, our use case requires *syntactical* correctness, as a syntactically invalid rule cannot be evaluated. However, each validation failure incurs an economic cost associated with another attempt at generating syntactically correct code.

To account for this economic impact, we adapt the "cost-to-pass" framework proposed by Erol et al. Erol et al. (2025). The authors propose a metric that calculates the expected monetary cost to obtain a correct solution as defined as:

$$v(m, p) = \frac{C_m(p)}{R_m(p)} \tag{2}$$

is the cost per attempt, and $R_m(p)$ is the success rate of that attempt.

We adapt this framework to fit our goal of minimizing the number of turns required to achieve syntactic correctness. Here, we define the success rate $R_m(p)$, the *pass@1* rate of our MQL validator, and the cost per attempt $C_m(p)$ as the cost of generating new MQL logic for that attempt.

In our testing, we apply this metric to help convey the real-world cost of autonomously generating valid rules. This is calculated as:

$$\text{Total Cost} = C_m(p) \times k \tag{3}$$

where k is the number of attempts required for the agent to generate MQL to pass the validation step. By calculating both the *pass@k* rate and the associated cost per generation, we can provide organizations with a metric that conveys the economic impacts of autonomously created security detections.

4.3. Robustness of Query

We develop a heuristic to measure the robustness of human-generated and LLM-generated email security rules, inspired by the defensive programming techniques in RobGen(Li et al., 2025), but adapted to address the unique challenges of detection engineering. While Li et al. determines robustness by examining the absence of conditional logic and improper error handling in generated code, our approach involves evaluating how susceptible detection rules are to adversarial evasion attempts. Recent work on LLM evaluation in cybersecurity contexts, such as CWEval (Peng et al., 2025), has shown that measuring code robustness, encompassing both functionality and security (e.g. resistance to vulnerabilities), is crucial to mitigating operational risks when deploying LLM-generated in production environments. In (Peng et al., 2025) and (Fu et al., 2024), the researchers introduce metrics to evaluate both the functionality and the security of the generated code.

These guiding principles apply to our use case (e.g., rule-based security detections), where defenders are in an active cat-and-mouse game with attackers who are actively attempting to evade these detectors through low-effort bypasses.

We assume that direct string matches on fields such as IP addresses, domains, or hash values indicate brittleness, as these patterns may be bypassed through trivial evasions by attackers. Conversely, we view behavioral indicators (sender prevalence, domain reputation scores, natural language understanding) and fuzzy matching as signs of robustness, as they present generalizable solutions to variations in attack patterns.

To quantify this, we compute a brittleness score $B \in [0, 100]$ that derives the score from the ratio of rewards (for robust patterns) to penalties (for brittle patterns). This ratio is mapped to a 0-100 scale using a logistic function. The formula is:

$$B = \frac{100}{1 + e^{k\left(\frac{R}{P} - x_0\right)}} \tag{4}$$

This metric accounts for the challenge all rule-based detections face in balancing brittle logic (*to avoid FPs*) and robust logic (*to avoid adversarial evasions*).

The robustness score is then:

$$\text{Robustness} = \frac{100 - B}{100} = 1 - \frac{B}{100} \tag{5}$$

Our approach differs from previous efforts to define code robustness metrics that focus on input validation and exception handling, because as DefenderBench demonstrates, the security domain requires robustness that accounts for resilience against adversarial evasions rather than accidental errors.(Zhang et al., 2025a)

5. Experimental Setup

Our goal is to illustrate and exercise the measurement of LLM-generated cybersecurity rule quality using held-out rules, for which we use ADÉ and the corpus of emails shared with Sublime Security as an illustrative example.

Table 1: List of Detectors Tested

Rule Name	Attack Type	Detection Method	TTPs
AFF from freemail/sus TLD	BEC/Fraud	Header, Content, Sender	Social eng.
Callback Phishing via PDF	Callback Scam	File Analysis, Exif Analysis	OOB pivot
Embedded JS in SVG	Malware	File, Sender Analysis	Scripting
EML w/ JS in SVG	Phish/Malware	File, Sender Analysis	Scripting/Evasion
HTML smuggling w/ atob	Malware	HTML analysis	HTML smuggling
BEC w/ Reply-to mismatch	BEC/Fraud	Header, Content, Sender	Evasion
Coinbase impersonation	Cred. Phishing	Header, Content, Sender	Brand Imp
DocuSign image lure	Cred. Phishing	Computer Vision, Content	Brand Imp
Extortion/sextortion	Extortion	Content, Header Analysis	Social eng.
Fake voicemail (untrusted)	Cred. Phishing	Content, URL Analysis	Social eng.
HTML smuggling	Cred. Phishing	HTML analysis	HTML smuggling
MS Dynamics 365 phishing	Cred. Phishing	File Analysis, OCR	Evasion
Open Redirect: Google /url	Cred. Phishing	Header, File Analysis	Evasion
QR Code w/ sus indicators	Cred. Phishing	Computer Vision	QR/Social eng.
Scam: Piano Giveaway	BEC/Fraud	Content Analysis	Social eng.
Spam: Fake photo share	Spam	Content, Sender Analysis	Social eng.
Explicit Google Group	Spam	Content, Sender Analysis	Social eng.
Suspected Lookalike domain	BEC/Fraud	Content, Sender Analysis	Evasion

5.1. Scenario

We use ADÉ, described in Section 3 as the tool to illustrate our evaluation framework. The primary scenario is a holdout validation process designed to compare ADÉ to a human baseline. We have curated a set of human written rules, given in table 1 designed to cover a broad range of tactics, techniques, and procedures (TTPs) from Sublime Security's production core feed (Sublime Core Feed). We then presented ADÉ with a TP, also human curated to be representative, that would have been caught by the held-out human written rule. We then measure the effectiveness of the ADÉ generated rule in comparison to the human written held-out rule.

5.2. Dataset

We are able to test ADÉ's performance by comparing to held out rules from the Sublime Security Core Feed, removing rules that ADÉ knows about one at a time and asking it to develop rules for examples of messages that would flag with the held out rule.

We chose a static list of rules to hold out in all runs of our test, with the goal of choosing rules that are likely to have been flagged with messages in our dataset and which represent a variety of TTPs and attacker goals. The rules chosen are listed in Table 1.

Our test environment consists of approximately $45,000$ email messages that have been submitted to Sublime Security for analysis over a 14 day period. For this purpose, we extended our labels by hand labeling the relevant emails as malicious or benign, augmenting our preexisting labeled data set with additional relevant labels.

6. Results

Here we examine the results of our measurement framework comparing rules from the Sublime Security core feed and those created by ADÉ in response to a single instance of the core feed rule, as chosen by a human expert to be representative. Of course, this evaluation is somewhat apples to oranges, since we tell ADÉ to address a single malicious email, but the expert humans who wrote the core rules have the goal of addressing a broad behavior based on a large collection of data. The complete evaluation results are given in tables 2, 3, and 4.

Table 2: Performance Metrics for Human-Generated Rules

Name	Hits	TPs	FPs	Unique TPs	Score
Callback solicitation via pdf file	765	756	9	747	0.982
EML w/ Javascript in SVG File	276	276	0	274	0.996
HTML smuggling with atob	16	14	2	12	0.813
BEC w/ Reply-to mismatch	1558	1557	1	793	0.754
Brand impersonation: Coinbase	35	33	2	31	0.914
Fake voicemail notification	344	304	40	264	0.826
MSFT Dynamics 365 form phish	57	57	0	24	0.711
Scam: Piano Giveaway	100	99	1	57	0.780
Spam: Fake photo share	240	231	9	208	0.913
Explicit Google Group Invite	10	10	0	10	1.000
Suspected Lookalike domain	10	8	2	8	0.800

Table 3: Performance Metrics for ADÉ-Generated Rules

Name	Hits	TPs	FPs	Unique TPs	Score
Callback solicitation via pdf file	31	31	0	23	0.871
EML w/ Javascript in SVG File	239	239	0	238	0.998
HTML smuggling with atob	4	4	0	1	0.625
BEC w/ Reply-to mismatch	35	35	0	4	0.557
Brand impersonation: Coinbase	132	116	16	75	0.723
Fake voicemail notification	6	6	0	5	0.917
MSFT Dynamics 365 form phish	56	56	0	24	0.714
Scam: Piano Giveaway	38	38	0	18	0.737
Spam: Fake photo share	202	202	0	187	0.963
Explicit Google Group Invite	7	7	0	7	1.000
Suspected Lookalike domain	6	6	0	6	1.000

Table 4: Brittleness and Cost Comparison

Rule Name	Brittleness (ADÉ)	Brittleness (Human)	Cost (ADÉ $)	ADÉ pass@k
Callback solicitation via pdf file	69.8	68.3	3.11	2
EML w/ Javascript in SVG File	68.2	72.3	5.13	3
HTML smuggling with atob	68.0	69.9	3.48	3
BEC w/ Reply-to mismatch	66.6	66.4	2.07	2
Brand impersonation: Coinbase	71.7	59.8	2.54	2
Fake voicemail notification	68.9	71.7	2.55	2
MSFT Dynamics 365 form phish	67.9	66.3	1.51	1
Scam: Piano Giveaway	70.6	72.5	2.07	2
Spam: Fake photo share	66.5	70.6	2.41	2
Explicit Google Group Invite	70.6	70.1	1.65	1
Suspected Lookalike domain	71.2	69.1	1.51	1

As one might expect from the constraints of ADÉ's design, the rules that it finds are much more narrowly tailored, finding both fewer overall messages and fewer known TPs, but also many fewer known FPs. This makes sense in that ADÉ is given a single message to work from, rather than the corpus of messages and idea about a set of behaviors to catch that humans use. Thus, ADÉ has a narrower view of the universe of malicious messages one might try to catch with this rule than the human who wrote the human rule. The fundamental security problem that "you don't know what you don't know" very much applies to ADÉ as well, since it gets only some of the problem set for humans to solve.

One result to note is that the pass@k metric correlates quite closely with the actual dollar cost of running the LLM to produce the result, as expected, see Figure 2.

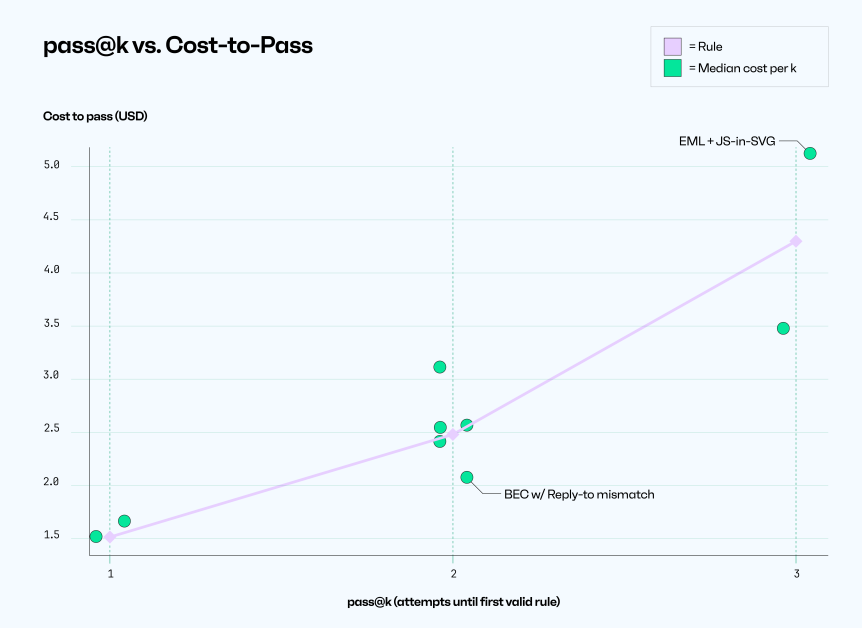

Figure 2: Cost of running the LLM as a function of *pass@k*.

ADÉ and humans are quite comparable in the brittleness of the rules they write. Security rules are, by their very nature, somewhat brittle, with difficult trade-offs about TPs, FPs, and FNs, suggesting that the base LLM is able to combine the information and tools it is given about query quality to produce high-quality queries.

One additional interesting discovery that we made as we compared ADÉ's rules with the human written rules is that ADÉ is much more inclined to write a lot of helpful comments than most human authors. We provide in Figure 3 a side-by-side view of a human written rule and the rule ADÉ wrote to illustrate the differences described here both in the breath of rules and the comments provided.

7. Conclusions and Future Work

There are two main branches of future work that we intend to pursue with the framework for evaluating security rules given here: further developing the framework and using the framework to further develop and refine ADÉ.

The hunt results of true and false positives reveal that ADÉ tends to find fewer results than the human-generated held-out rules, even though it makes quite robust rules and is reasonably economically efficient. This is not surprising, given the design of ADÉ to work on a single malicious email at a time. We can use the measurement framework described here for iterative improvements to ADÉ, for example finding an effective method to broaden ADÉ's scope to build detections that rival the recall of human written detections.

In addition, we would like to expand the testing framework described here in a number of ways. In real life, email security systems are expected to filter spam and graymail (potentially unwanted bulk mailing list messages) from mailboxes, in addition to filtering malicious messages. It would be useful to expand this framework to other categories that are of interest with a multimodal version of the framework. Furthermore, it would be helpful to include adversarial robustness tests in addition to the current evaluations of the rules, where we evaluate the generated rules against a number of automutated threat variants in addition to looking for static signs of adversarial robustness. Adversarial robustness tests likely would further help us improve ADÉ in making broader rules in the way that humans currently do.

Figure 3: Comparison of ADÉ-generated and Human-generated rules for detecting HTML smuggling

ADÉ Generated Rule

```
type.inbound

// SECTION 1: EML attachment with SVG content
and any(attachments,
  .content_type == "message/rfc822"
  and strings.icontains(file.parse_text(.).text, ".svg")
)

// SECTION 2: Base64 encoded JavaScript
// obfuscation patterns
and any(attachments,
  .content_type == "message/rfc822"
  and any(beta.scan_base64(file.parse_text(.).text),
    (
      strings.ilike(.,
        "*atob*",
        "*eval*",
        "*fromCharCode*"
      )
      or strings.ilike(.,
        "*window.location*",
        "*document.location*"
      )
      or regex.icontains(.,
        "parseInt.*charCodeAt")
    )
  )
)

// SECTION 3: Recipient targeting indicator
and any(attachments,
  .content_type == "message/rfc822"
  and any(recipients.to,
    strings.icontains(file.parse_text(..).text,
      .email.email
    )
    and .email.domain.valid
  )
)

// SECTION 4: Sender validation
and (
  profile.by_sender().prevalence in (
    "new",
    "outlier"
  )
  or not profile.by_sender().solicited
  or not headers.auth_summary.dmarc.pass
)
```

Human-generated Rule

```
type.inbound
and any(attachments,
  (.content_type == "message/rfc822"
   or .file_extension =~ "eml")
  and (
    any(file.parse_eml(.).attachments,
      .file_extension in~ ("svg", "svgz")
      and (
        (
          strings.ilike(file.parse_text(.).text,
            "*onload*",
            "*window.location.href*",
            "*onerror*",
            "*CDATA*",
            "*<script*",
            "*</script*",
            "*atob*",
            '*location.assign*',
            '*decodeURIComponent*'
          )
          or regex.icontains(file.parse_text(.).text,
            '<iframe[^\>]+src\s*=\s*\"data:[^\;]+;base64,'
          )
          or any(beta.scan_base64(file.parse_text(.).text),
            strings.ilike(.,
              "*onload*",
              "*window.location.href*",
              "*onerror*",
              "*CDATA*",
              "*<script*",
              "*</script*",
              "*atob*",
              '*location.assign*',
              '*decodeURIComponent*'
            )
          )
        )
        or // Additional sender logic...
      )
    )
  )
)
```

References

Mert Cemri, Melissa Z. Pan, Shuyi Yang, Lakshya A. Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. Why do multi-agent llm systems fail? *arXiv preprint 2503.13657*, 2025.

Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, Wei Ye, Yue Zhang, Yi Chang, Philip S. Yu, Qiang Yang, and Xing Xie. A survey on evaluation of large language models. *ACM Trans. Intell. Syst. Technol.*, 15(3), March 2024. ISSN 2157-6904. doi: 10.1145/3641289. URL https://doi.org/10.1145/3641289.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *arXiv preprint 2107.03374*, 2021.

Yuxiao Chen, Jingzheng Wu, Xiang Ling, Changjiang Li, Zhiqing Rui, Tianyue Luo, and Yanjun Wu. When large language models confront repository-level automatic program repair: How well they done? *arXiv preprint arXiv:2403.00448*, 2024.

Ads Dawson, Rob Mulla, Nick Landers, and Shane Caldwell. Airtbench: Measuring autonomous ai red teaming capabilities in language models. *arXiv preprint arXiv:2506.14682*, 2025.

Connor Dilgren, Purva Chiniya, Luke Griffith, Yu Ding, and Yizheng Chen. Secrepobench: Benchmarking llms for secure code generation in real-world repositories. *arXiv preprint arXiv:2504.21205*, 2025.

Mehmet Hamza Erol, Batu El, Mirac Suzgun, Mert Yuksekgonul, and James Zou. Cost-of-pass: An economic framework for evaluating language models. *arXiv preprint arXiv:2504.13359*, 2025.

Richard Fang, Rohan Bindu, Akul Gupta, Qiusi Zhan, and Daniel Kang. Llm agents can autonomously hack websites. *arXiv preprint arXiv:2402.06664*, 2024.

Yanjun Fu, Ethan Baker, Yu Ding, and Yizheng Chen. Constrained decoding for secure code generation. *arXiv preprint arXiv:2405.00218*, 2024.

Yuchuan Fu, Xiaohan Yuan, and Dongxia Wang. Ras-eval: A comprehensive benchmark for security evaluation of llm agents in real-world environments. *arXiv preprint 2506.15253*, 2025.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

Sayash Kapoor, Benedikt Ströbl, Zachary S. Siegel, Nitya Nadgir, and Arvind Narayanan. Ai agents that matter. *arXiv preprint arXiv:2407.01502*, 2024. doi: 10.48550/arXiv.2407.01502.

Zike Li, Mingwei Liu, Anji Li, Kaifeng He, Yanlin Wang, Xin Peng, and Zibin Zheng. Enhancing the robustness of llm-generated code: Empirical study and framework. *arXiv preprint 2503.20197*, 2025.

Aleksandar Milenkoski, Marco Vieira, Samuel Kounev, Alberto Avritzer, and Bryan D. Payne. Evaluating computer intrusion detection systems: A survey of common practices. *ACM Comput. Surv.*, 48(1), September 2015. ISSN 0360-0300. doi: 10.1145/2808691. URL https://doi.org/10.1145/2808691.

Samuel Miserendino, Michele Wang, Tejal Patwardhan, and Johannes Heidecke. Swe-lancer: Can frontier llms earn 1 million from real-world freelance software engineering? *arXiv preprint arXiv:2502.12115*, 2025.

Samuele Pasini, Jinhan Kim, Tommaso Aiello, Rocio Cabrera Lozoya, Antonino Sabetta, and Paolo Tonella. Evaluating and improving the robustness of security attack detectors generated by llms. *arXiv preprint arXiv:2411.18216*, 2024.

Jinjun Peng, Leyi Cui, Kele Huang, Junfeng Yang, and Baishakhi Ray. Cweval: Outcome-driven evaluation on functionality and security of llm code generation. In *2025 IEEE/ACM International Workshop on Large Language Models for Code (LLM4Code)*, pages 33–40. IEEE, 2025.

Ethan M. Rudd, Christopher Andrews, and Philip Tully. A practical guide for evaluating llms and llm-reliant systems. *arXiv preprint arXiv:2506.13023*, 2025.

Sublime Core Feed. https://sublime.security/feeds/core/.

Shiqi Wang, Zheng Li, Haifeng Qian, Chenghao Yang, Zijian Wang, Mingyue Shang, Varun Kumar, Samson Tan, Baishakhi Ray, Parminder Bhatia, et al. Recode: Robustness evaluation of code generation models. *arXiv preprint arXiv:2212.10264*, 2022.

Chiyu Zhang, Marc-Alexandre Cote, Michael Albada, Anush Sankaran, Jack W Stokes, Tong Wang, Amir Abdi, William Blum, and Muhammad Abdul-Mageed. Defenderbench: A toolkit for evaluating language agents in cybersecurity environments. *arXiv preprint arXiv:2506.00739*, 2025a.

Linghao Zhang, Shilin He, Chaoyun Zhang, Yu Kang, Bowen Li, Chengxing Xie, Junhao Wang, Maoquan Wang, Yufan Huang, Shengyu Fu, Elsie Nallipogu, Qingwei Lin, Yingnong Dang, Saravan Rajmohan, and Dongmei Zhang. Swe-bench goes live! *arXiv preprint 2505.23419*, 2025b.