

WAFGen: LLM Agents for IDS and Multi-Cloud WAF Rules Generation

Anes Abdennebi

`anes.abdennebi.1@ens.etsmtl.ca`

École de Technologie Supérieure

Nadjia Kara

École de Technologie Supérieure

Hakima Ould-Slimane

Université du Québec à Trois-Rivières

Research Article

Keywords: LLM agents, large language models, WAF, cloud security, IDS, rule generation, AWS WAFv2

Posted Date: October 23rd, 2026

DOI: <https://doi.org/10.21203/rs.3.rs-11273238/v1>

License: © ⓘ This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

Additional Declarations: No competing interests reported.

WAFGen: LLM Agents for IDS and Multi-Cloud WAF Rules Generation

Anes Abdennebi^{1*}, Nadja Kara¹ and Hakima Ould Slimane²

¹Department of Software Engineering and IT, École de Technologie Supérieure, Montreal, Canada.

²Department of Mathematics and Computer Science, Université du Québec à Trois-Rivières, Trois-Rivières, Canada.

*Corresponding author(s). E-mail(s): anes.abdennebi.1@ens.etsmtl.ca;

Abstract

Cloud security operations increasingly depend on accurate, provider-specific network intrusion detection and web application firewall rules, yet manual rule authoring remains, slow and costly against a dynamic threat landscape. This paper presents WAFGEN, a multi-LLM fine-tuning pipeline for the automatic generation of Intrusion Detection System (IDS) and Web Application Firewall (WAF) rules targeting two major IDS frameworks and three leading cloud providers: Amazon Web Services (AWS), Microsoft Azure, and Cloudflare. WAFGEN introduces two training families (SINGLE-PLATFORM and MULTI-PLATFORM adapters) trained on open-source threat corpora. Fine-tuned models achieve over 90% syntactic validity across all platforms, with MISTRAL-7B reaching 99.6% and CODELLAMA-7B reaching 94% under live API deployment; multi-platform adapters match or exceed their single-platform counterparts across all evaluated metrics, including BLEU, chrF, and F1-score, demonstrating strong cross-provider generalization. WAFGEN-generated rules further exhibit a mean AWS Web ACL Capacity Unit (WCU) cost of 10.4–25.0 WCU, up to 28× reduction relative to the 700 WCU AWS managed rule set baseline, enabling pre-deployment cost estimation at rule-generation time. These results position WAFGEN as both an LLM-as-a-Service capability for cloud providers and a locally deployable tool that allows end users to adapt to evolving threats at zero marginal cost within a controlled WCU budget.

Keywords: LLM agents, large language models, WAF, cloud security, IDS, rule generation, AWS WAFv2

1 Introduction

Modern communication networks increasingly span cloud-native applications, edge gateways, service meshes, and hybrid enterprise backbones. This distribution improves agility, yet it expands the attack surface that security teams must observe

and control. Intrusion Detection Systems (IDS), Extended Detection and Response (XDR) platforms, and web application firewalls remain critical for this objective because they transform threat intelligence into executable detection logic. Their effectiveness depends on timely and precise rules

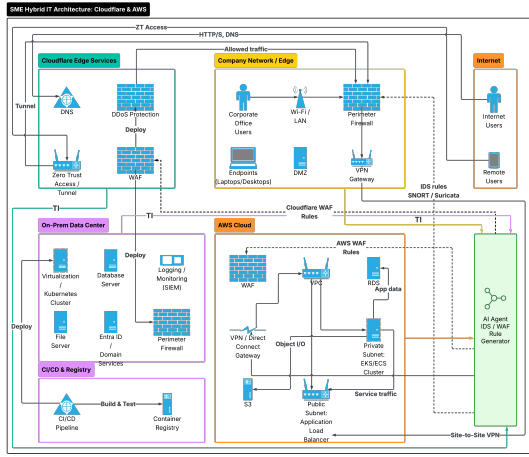


Fig. 1 WAFGen block diagram for a hybrid enterprise deployment. The AI rule-generation agent sits between security knowledge sources and enforcement points, producing Snort and Suricata IDS rules for on-premise or edge sensors and provider-specific WAF rules for Cloudflare, AWS, and Azure. TI stands for Threat Intelligence.

that match malicious behaviour while minimizing false positives on legitimate traffic.

Rule authoring is a serious operational gap. Security engineers must translate indicators, exploit descriptions, and anomaly patterns into target-specific syntax. Snort and Suricata rules combine a header with action, protocol, address scope, direction, and ports, then attach options such as messages, content matchers, regular expressions, flow state, metadata, identifiers, and revisions. WAF rules differ even more across cloud providers. AWS WAFv2 uses JSON statements such as regex match, byte match, rate based, and managed rule group statements with visibility configuration and rule capacity accounting. Cloudflare Ruleset Engine rules use an expression language over HTTP fields with actions such as block, challenge, log, skip, or execute. Azure Application Gateway WAF policies encode managed rule sets and custom match conditions with transforms, operators, priorities, and actions. These structures are expressive, but their complexity makes manual conversion from threat context to deployable rules slow, expensive, and error-prone.

Web Application Firewalls serve as a fundamental mechanism for enforcing security policies at the application layer, providing rule-based filtering against a broad spectrum of attacks including SQL injection, Cross-Site Scripting (XSS), and automated bot-driven threats [1] [2]. Unlike

network-layer firewalls, WAFs inspect the content of HTTP/HTTPS transactions, enabling fine-grained control over request payloads, headers, cookies, and URI paths. The three dominant cloud-based WAF platforms, AWS WAF, Microsoft Azure WAF, and Cloudflare WAF, share a common functional baseline yet differ substantially in their architectural positioning and optimization targets: Cloudflare operates as an edge-native solution distributed across its global anycast network, AWS WAF integrates within cloud-hosted infrastructure through constructs such as Web ACLs, and Azure WAF is deeply coupled with application delivery components including Application Gateway and Azure Front Door [3–5].

A primary deployment scenario shared across all three platforms is protection against the vulnerabilities enumerated in the OWASP Top 10, which catalogs the most critical web application security risks [1]. Each platform exposes managed rulesets that are maintained and updated by the respective provider. AWS Managed Rules offer pre-built protection groups targeting Linux environments, SQL databases, and known malware signatures, while Azure’s Default Rule Set (DRS) is integrated with Microsoft Threat Intelligence feeds, and Cloudflare Managed Rulesets provide zero-day patch propagation across the global edge network without requiring manual intervention by operators [3–5]. This managed model reduces the operational burden on security teams, particularly in environments where dedicated security expertise is limited, though it introduces a dependency on the provider’s update cadence and rule transparency.

Beyond vulnerability mitigation, WAF platforms support geo-fencing and access control policies that restrict or permit traffic based on geographic origin, Autonomous System Number (ASN), or IP address. Common use cases include regulatory compliance enforcement through country-level traffic blocking, and the protection of administrative interfaces by whitelisting known corporate IP ranges [3–5]. On the other hand, bot management and rate limiting represent an increasingly critical WAF function in response to the proliferation of automated threats such as credential stuffing, content scraping, and distributed brute-force attacks [2, 6]. AWS WAF applies rate-based rules over configurable time

windows, Azure WAF employs rate-limiting custom rules to suppress endpoint traffic spikes, and Cloudflare’s dedicated Bot Management layer analyzes behavioural signals and assigns bot scores to distinguish legitimate crawlers from malicious automation, issuing interactive challenges rather than unconditional drops.

Finally, all three platforms expose mechanisms for encoding custom application logic within WAF rules, enabling operators to inspect arbitrary HTTP attributes, including URI paths, query strings, request bodies, and custom headers, and to trigger differentiated actions such as blocking, logging, or issuing CAPTCHA challenges. AWS WAF implements this through Web ACL rule statements with URI and header inspection conditions. Azure WAF supports compound logical conditions (e.g., conjunctions of URL pattern matching and IP set membership), and Cloudflare provides a proprietary Rules Language that evaluates structured fields including cookies, JWT tokens, and custom headers to enforce application-specific access policies [3–5]. The expressiveness and deployment flexibility of these rule languages determine how effectively a WAF can be adapted to a unique and dynamic threat surface of a given application, and represent a key differentiator among platforms in enterprise deployment scenarios.

Large language models can reduce this burden by mapping natural security context to executable rules. However, general-purpose models hallucinate unsupported operators, mix provider fields, omit mandatory rule identifiers, and generate broad matchers that increase false positives. WAFGen addresses this problem with specialized rule generation agents trained on curated Snort, Suricata, and multi-cloud WAF examples. The system builds on LI-NIDS for LLM-based NIDS rule generation and governance-oriented multi-agent security architectures by adding cloud WAF targets and provider-aware validation [7, 8].

The contributions of this paper are threefold. **First**, we introduce a unified corpus and prompt format for five rule targets across IDS and cloud WAF ecosystems, with explicit WAF augmentation to address scarce provider-native examples. **Second**, we describe a QLoRA training and validation workflow that separates IDS-specialized adapters from WAF-capable adapters because their grammars, validators, deployment targets, and

failure modes differ. **Third**, and most importantly, we present the first live deployment evaluation of locally-adapted open LLMs against real AWS WAFv2, Cloudflare, and Azure WAF APIs: MISTRAL-7B and CODELLAMA-7B exceed 90% acceptance on AWS and Azure, all models produce rules with mean WCU of 10.4–25.0 (against a 1,500 WCU Web ACL default ceiling and a 700 WCU cost for the AWS managed common rule set), and WAFGen delivers per-rule WCU feedback through `wafv2:CheckCapacity` before any WebACL is created. The code, used and generated datasets are publicly available¹.

The remainder of this paper is organized as follows. Section 2 reviews rule generation and LLM security automation. Section 3 presents data curation, training, validation, and deployment. Section 4 reports measured results including live deployment metrics. Section 5 concludes with limitations and future work.

2 Related Work

Recent work since 2024 has accelerated the use of LLMs in cyber operations, including security question answering [9–11], alert triage [12, 13], code vulnerability analysis [14, 15], malware reasoning [16], and autonomous security agents [17]. CyberSecEval 2 evaluates LLM safety and cybersecurity capability across offensive and defensive tasks, showing that model specialization and guardrails are necessary for reliable cyber use cases [10]. Agentic security platforms build on this direction by decomposing workflows into planning, retrieval, validation, and execution steps. Governance-aware multi-agent security work motivates this design, while WAFGen focuses the agentic loop on rule synthesis and provider validation [8].

A key enabler for grounding LLM-generated security artefacts in current threat intelligence is Retrieval-Augmented Generation (RAG). RAG-Intel demonstrates that a hybrid RAG architecture combining dense retrieval, BM25 sparse search, and contextual reranking allows LLMs to answer attack investigation queries with substantially higher factual accuracy than vanilla zero-shot prompting across 339 diverse cybersecurity tasks [18]. Domain-specific extensions such as

¹<https://github.com/RoronoaZ/WAFGen>

CyKG-RAG build knowledge graphs from structured threat databases (e.g. MITRE ATT&CK) to enrich retrievers with semantic relationships between attack techniques [18]. Complementing this, automated multi-agent pipelines such as CVE-Genie and the framework of Lotfi et al. address the upstream problem of converting raw CVE disclosures into verifiable, containerised exploits by augmenting incomplete NVD descriptions via RAG, synthesising attack environments, and iteratively validating results [19, 20]. These RAG and vulnerability-reproduction pipelines are natural feeder stages for WAFGen: they produce the structured threat context that conditions rule generation and provide ground-truth traffic against which generated rules can be replayed and validated.

ML and DL based IDS research continues to improve traffic classification with transformers, graph neural networks, autoencoders, and federated learning. Hybrid frameworks combining GNNs and Transformer architectures have shown strong results, fusing spatial and temporal traffic representations to improve robustness against diverse attack types in IoT and critical infrastructure environments [21, 22]. Self-supervised approaches such as GraphIDS jointly train a GNN encoder and a Transformer-based masked autoencoder on benign NetFlow traffic, flagging anomalies via reconstruction error without relying on labeled attack samples [23]. Privacy-preserving methods like FL-TIDS combine unsupervised autoencoder-based detection with federated learning to reduce dependence on labeled samples while protecting distributed data sources [24]. These approaches often optimize detection accuracy over packet or flow datasets, but they do not automatically produce operational Snort or Suricata rules that can be reviewed, deployed, and audited by analysts. WAFGen is complementary because it generates human-readable rules that can seed IDS and XDR pipelines, then be tested against traffic replays.

Signature and WAF communities have long relied on expert-maintained rule repositories such as Snort community rules, Emerging Threats Open Suricata rules, and OWASP Core Rule Set [25–27]. These projects provide high-quality baselines, yet cloud provider rule languages diverge rapidly. AWS WAF, Cloudflare Ruleset Engine, and Azure WAF expose different match variables, action

vocabularies, capacity constraints, and deployment templates [3–5].

Rather than replacing these rule repositories, recent work has explored augmenting them with learned models. ModSec-Learn treats OWASP CRS rules as input features to a machine-learning classifier, allowing the model to tune the contribution of each rule to detection decisions and adapt severity thresholds to the specific web application being protected [28]. At the rule-generation level, RuleMaster+ fine-tunes a large language model on a curated instruction dataset to automate IDS rule writing. Even for previously unseen attacks, the system generates effective rules from proof-of-concept exploit descriptions and outperforms off-the-shelf LLMs on rule generation benchmarks [29]. RuleX-ploit similarly targets Suricata, using structured prompt engineering over GPT-4o to produce syntactically valid rules directly from exploit code, achieving full syntactic validity and a 76.67% malicious-traffic detection rate in empirical evaluation [30]. Existing converters and generators cover narrow cases (a single rule engine, a fixed exploit format, or a specific application context) while WAFGen learns a target-conditioned generation function that can draft platform-native rules from the same threat context and extends to cover multiple heterogeneous cloud WAF deployments.

More recent work has cast rule generation as a full agentic workflow rather than a single-pass generation task. Hex2Sign showed that LLMs fine-tuned on hexadecimal payloads extracted from honeypot PCAPs can synthesize human-readable Suricata signatures directly from raw binary traffic, achieving a BERTScore above 96% [31]. Building on this, dedicated multi-source LLM agents were proposed to simultaneously generate rules and produce generalized variants capable of covering related attack families [32]. FALCON extends the agentic design to the full intelligence cycle: it autonomously mines real-time CTI feeds, extracts attack patterns, and passes candidate rules through a multi-phased validation pipeline before deployment, achieving approximately 95% accuracy across both Snort and YARA targets with strong inter-analyst agreement [33]. GRIDAI further addresses rule-set maintenance by coupling generation with repair, a four-agent collaboration first classifies whether an incoming sample represents a new attack or a variant of a known

threat, then either generates a new rule or surgically updates an existing signature, mitigating rule-set bloat while suppressing LLM hallucinations through live engine validation [34]. RulePilot introduces an intermediate representation (IR) that decouples the LLM from the syntactic particularities of individual rule grammars, enabling a single agent to create or convert rules across detection engines reaching a significant improvement ratio in textual fidelity over baseline models and demonstrably better execution accuracy on real-world traffic [35].

Despite this rapid progress, all of these systems share a common constraint: they are hardwired to a single detection engine grammar (e.g., Snort, Suricata, or YARA) and produce no artifact for cloud-provider WAF deployments. None of them account for the divergence in match variables, action vocabularies, capacity quotas, and Terraform or CloudFormation deployment templates that separates AWS WAF, Cloudflare Ruleset Engine, and Azure WAF from each other. WAFGen’s target-conditioned generation function addresses precisely this gap: given a threat context, a single inference pass produces a provider-specific rule set for whichever cloud target is specified, bridging the IDS rule generation literature and the multi-cloud WAF deployment challenge for the first time.

At the broader cloud security automation level, Toprani and Madisetti demonstrate that multi-agent LLM systems augmented with RAG can reason over Infrastructure-as-Code (IaC) templates, detect context-specific misconfigurations overlooked by static analysis tools, and propose remediations with an 85% detection rate across diverse cloud frameworks [36]. This confirms that LLM agents can operate reliably on platform-specific configuration artefacts at cloud scale, providing complementary evidence for the design choices in WAFGen. While Toprani et al. target configuration correctness and compliance, WAFGen targets threat-driven rule synthesis, together, these directions outline a future in which the full lifecycle of cloud security policy (from vulnerability disclosure through rule generation to deployment-time validation) is driven by co-operating LLM agents.

3 Methodology

WAFGen addresses the following generation problem: given a natural-language threat description x and a target platform identifier $p \in \{Snort, Suricata, AWS, Cloudflare, Azure\}$, produce a syntactically valid, operationally deployable rule $\hat{y}$ that conforms to the grammar $\mathcal{G}_p$ of platform p . The five platforms span two fundamentally different structural families, text-based IDS rule languages (Snort, Suricata) and JSON-schema-driven cloud WAF policies (AWS, Cloudflare, Azure), whose differences in action vocabularies, namespaces, capacity units, and deployment templates are summarized in Table 2.

This work places particular emphasis on WAF rule generation for the three leading cloud providers. Unlike IDS rules that can be evaluated offline against PCAPs, WAF rules must be accepted and provisioned by live provider APIs before getting deployed to any workload. Additionally, AWS WAF imposes a hard *Web ACL Capacity Unit* (WCU) budget: every statement type carries a fixed WCU cost that is validated at rule creation time via the `wafv2:CheckCapacity` API call. A rule or combination of rules that exceed the capacity budget (5000 WCU/Web ACL) are rejected at deployment regardless of its semantic correctness. WAFGen therefore enables WCU awareness directly into its evaluation pipeline, giving operators a pre-deployment cost estimate before any rule is submitted to the provider, a capability, which is to the best of our knowledge, absent from all prior IDS and WAF rule-generation systems in the literature.

3.1 Corpus Design and Data Curation

3.1.1 Balanced Cross-Platform Corpus

Each platform is allocated an equal share of the training budget to prevent the collective model from developing a bias toward the most common rule family. As shown in Table 1, the final corpus contains 5,000 labelled instances per platform (4,250 for training, 500 for validation, and 250 for held-out testing), giving 25,000 instances in total across the five targets.

3.1.2 Curation Pipeline

Raw rule material was collected from three complementary sources: (i) official community repositories (Snort Community Rules, Emerging Threats Open, OWASP CRS [25–27]), (ii) provider documentation examples and publicly available CloudFormation / ARM / Terraform WAF templates [3–5], and (iii) synthetic instances generated by a seed LLM under strict structural constraints and subsequently verified by platform-specific validators. All instances underwent a three-stage quality filter:

1. **Structural validation.** Each rule is parsed by the relevant native validator—a Snort/Suricata engine for IDS rules, and the provider schema validator or a locally reproduced JSON Schema for WAF rules. Any instance that fails parsing is discarded.
2. **Operator-set audit.** For WAF JSON rules, every operator token (comparison predicates, transformation functions, action keywords) is checked against the finite operator set Ω_t of the target platform. Rules referencing undocumented or deprecated operators are removed.
3. **Semantic deduplication.** Near-duplicate pairs whose edit distance falls below a tuned threshold are collapsed to a single canonical representative, ensuring that no single attack pattern dominates the training distribution.

The prompt for each instance is a concise, human-readable threat description that a security analyst might type, deliberately phrased without any platform keyword so that the model must infer the correct grammar solely from the target identifier p injected at inference time. This design choice is what allows WAFGen to behave as a target-conditioned rule generator.

3.2 WAFGen Pipeline

Figure 1 shows the WAFGen placement in a hybrid SME architecture. The agent receives threat intelligence, vulnerability reports, alerts, analyst tickets, and CI artifacts, then drafts either Snort and Suricata signatures or provider-native WAF policies. Figure 2 show the validation and deployment flow of generated rules after being reviewed and versioned through controlled pipelines to IDS sensors, Cloudflare edge services, AWS WAF, or Azure WAF. Raw sources are downloaded or read from a local cache, normalized into target-specific records,

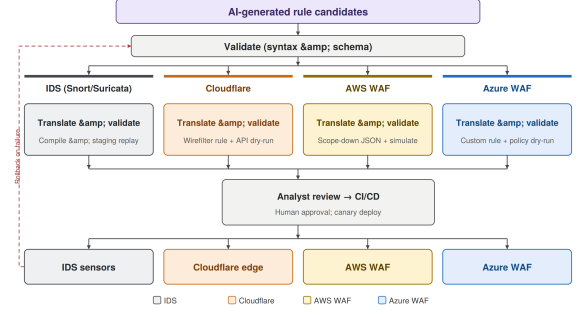


Fig. 2 WAFGen validation and deployment pipelines for Snort, Suricata, AWS, Cloudflare, and Azure.

Table 1 Balanced rule-generation corpus

Split	Snort	Suricata	AWS	Cloudflare	Azure
Train	4250	4250	4250	4250	4250
Validation	500	500	500	500	500
Test	250	250	250	250	250
Total	5000	5000	5000	5000	5000

Table 2 Rule structures targeted by WAFGen

Target	Main structure	Validation focus
Snort	Alert header plus options	Action, direction, message, SID, revision, payload matcher
Suricata	Snort-like rule with richer protocol keywords	Header, options, protocol fields, identifiers
AWS WAF	JSON rule with statement and action	Name, priority, statement, action or override action
Cloudflare WAF	JSON rule with expression and action	Expression language, supported actions, enabled flag
Azure WAF	JSON policy rule or managed rule set	Name, match conditions or managed rules, priority, action or

split into train, validation, and test partitions, fine-tuned with QLoRA, and evaluated by syntax, schema, format, and text metrics. Each record uses an instruction, a threat context, an output rule, a target format, a source tag, and a rule type.

3.3 Model Selection

Four pre-trained base models spanning two parameter-size classes were selected to assess the

trade-off between model capacity and deployment practicality:

- **Phi-3-mini** [37]: 3.8B parameters. Chosen as a representative of the compact instruction-following class that can run on a single consumer GPU, providing a lower bound on the compute required to achieve acceptable generation quality.
- **Qwen3-1.7B-Base** [38]: 1.7B parameters. The smallest model in the set, included to establish whether sub-2B models retain enough structured-output capacity to generate valid WAF JSON after fine-tuning.
- **CodeLlama-7B** [39]: 7B parameters, pre-trained on code corpora. Its code pretraining is directly relevant to JSON WAF rules and to the strict syntactic requirements of Snort/Suricata grammars.
- **Mistral-7B** [40]: 7B parameters, strong general-purpose baseline with demonstrated instruction-following fidelity. Provides a second 7B data point to separate code-specific from general capacity effects.

This selection enables an explicit comparison that is a research objective in its own right: whether compact 2–4B models can match 7B models on structured security rule generation after task-specific fine-tuning.

3.4 Collective vs. Per-Format Fine-Tuning

Each of the four base models is fine-tuned under two distinct paradigms, yielding eight fine-tuned checkpoints per paradigm (32 in total).

Collective Training - Multi-platform models.

The model receives the full 25,000-instance corpus. The target platform p is prepended to every prompt as a control token, making the generation function $p_\theta(\hat{y} \mid x, t)$ explicitly conditioned on the desired format. The hypothesis is that a single collective model can generalize across all five structural families given sufficient data. Later in the experiment section, we label this category as **Multi-platform models/adapters**.

Per-Format Training - Single-platform models.

Five separate fine-tuned instances of each base model are trained, one per platform, each seeing only the 5,000 instances allocated to its target. The platform token is omitted from the prompt since the model is always deployed for a single known target. This paradigm maximizes format specialization at the cost of requiring the operator to select the correct model at inference time. Later in the experiment section, we label this category as **Single-platform models/adapters**.

Hypothesis.

We hypothesize that *Single-platform* models will consistently outperform their *Multi-platform* counterparts across all structural and deployment metrics. The rationale is threefold: (i) the five grammars are mutually dissimilar enough that shared training introduces inter-format interference, except for the IDS and Suricata rule structures, their formats are similar, with few field differences. (ii) The JSON WAF family and the text-based IDS family impose conflicting tokenization patterns that compete during gradient updates, and (iii) per-format fine-tuning allows the model to overfit beneficially to the precise operator vocabulary and field-ordering constraints of a single target, reducing hallucinated tokens. Verifying or refuting this hypothesis is a primary experimental goal. A finding in favour of *Single-platform* also has a direct operational implication: organizations that deploy only one provider can use the lighter, specialized checkpoint rather than the larger collective model, reducing both inference cost and latency. We exclude the IDS family from the hypothesis evaluation experiments (*Multi* vs. *Single-platform* models) due to the above-mentioned first rationale in the stated hypothesis.

3.5 Training Algorithm

Let each sample be $x_i = (I_i, C_i, F_i, y_i)$, where I_i is the instruction, C_i is the threat context, F_i is the target format, and y_i is the reference rule. The agent learns a conditional distribution $p_\theta(y_i \mid$

I_i, C_i, F_i). Supervised fine-tuning minimizes token-level negative log likelihood

$$\mathcal{L}(\theta) = - \sum_i \sum_{t=1}^{T_i} \log p_\theta(y_{i,t} \mid I_i, C_i, F_i, y_{i,<t}). \quad (1)$$

Where T_i represents the number of tokens of the reference rule y_i for the i^{th} training sample. t is the current token in the reference rule being predicted during the training. For a linear layer W , the adapted weight is

$$W' = W + \frac{\alpha}{r} BA, \quad (2)$$

where A and B are trainable low-rank matrices, r is the rank, and α scales the update. WAFGen uses 4-bit NF4 quantization, double quantization, LoRA rank 8, LoRA dropout 0.05, cosine learning rate scheduling, gradient accumulation, and a per-family step budget: the primary IDS and WAF adapters described in this section are trained for **300 steps** over the 4,250-record per-platform training split, and the platform-specialization adapters introduced in Section 4.4 use **500 steps** to accommodate the larger 12,750-record training corpora used in both the single-v2 and multi-platform conditions.

We train two adapter families. The IDS family targets Snort and Suricata generation and includes PHI-3-MINI, MISTRAL-7B-Instruct, CODELLAMA-7B-Instruct, and QWEN3-1.7B-Base adapters trained on the IDS split. The WAF family targets AWS, Cloudflare, and Azure WAF using the same base model families. This separation is deliberate: IDS rules are line-oriented signatures with strict option syntax and traffic semantics, while WAF rules are provider-specific JSON or expression policies with schema, operator, priority, and capacity constraints.

3.6 Live Deployment Evaluation

The deployment path follows the review-before-deploy principle in Figure 1. IDS outputs are staged as Snort or Suricata rule files for future possible deployment on firewalls or intrusion detection systems, while WAF outputs are staged as Cloudflare expressions, AWS WAFv2 statements, or Azure WAF policy rules, kept for review before deploying them to their respective platforms.

WAFGen includes a production-stage live deployment evaluation tier, gated behind an explicit `--allow-live` flag (in terminal), that submits generated rules to real cloud provider APIs and records the outcomes. Each provider interaction is described below.

AWS WAFv2 — `wafv2:CheckCapacity`

For each generated AWS WAF rule, WAFGen calls the boto3 `wafv2.check_capacity()` method, supplying the exact generated rule JSON wrapped in a one-element `Rules` array. This is a read-only call that requires only the `wafv2:CheckCapacity` Identity and Access Management (IAM) permission and creates no cloud resources. The AWS evaluator returns the integer number of Web ACL Capacity Units (WCU) the rule would consume, allowing operators to know the cost of the rule before any WebACL is created. Round-trip latency is recorded per call. A second, optional `deploy_and_test()` path creates a temporary REGIONAL WebACL tagged `wafgen:eval`, confirms acceptance, measures provisioning latency, and deletes the WebACL immediately. This path requires the additional `wafv2:CreateWebACL`, `wafv2:GetWebACL`, and `wafv2>DeleteWebACL` permissions. All temporary resources are tagged and cleaned up automatically.

Cloudflare Rulesets API

For each generated Cloudflare WAF rule, WAFGen submits the rule with `"enabled": false` to a dedicated `wafgen-eval` custom ruleset on the operator's test zone. Since the rule is disabled, it is *never evaluated against live traffic*. The Cloudflare API responds with HTTP 200 on successful acceptance or HTTP 400 with structured error details for expression or schema failures. WAFGen records the HTTP status, round-trip latency, and any API error messages. If the operator's Cloudflare plan does not permit custom rulesets (HTTP 403), WAFGen automatically falls back to the expression-only validator, which accepts expressions for validation without creating any rule object.

Azure Application Gateway WAF

For each generated Azure WAF rule, WAFGen wraps the generated policy object in a full Application Firewall Policy² ARM envelope, provisions it in *Detection mode* (zero traffic impact) under the configured resource group using the Azure Management SDK `DefaultAzureCredential` chain, and records the ARM provisioning state and round-trip latency. All policies carry the tag `wafgen:eval` for easy identification. Once acceptance is confirmed, WAFGen deletes the policy. An ARM provisioning state of `Updating` is the expected synchronous acknowledgement for a successfully accepted policy, reflecting the ARM asynchronous convergence model where the control plane confirms acceptance before background resource allocation completes.

4 Experiments and Evaluation

All experiments were conducted on a single workstation running Ubuntu 22.04 LTS with Linux kernel 6.8.0, and Python 3.12.12. The hardware comprises an Intel processor (i7 12th Gen) with 20 CPU cores and 32 GB of RAM, while an NVIDIA RTX 4080 Super GPU provides CUDA training, acceleration, and testing for LLMs finetuning.

4.1 Metrics

The reported metrics are syntactic and schema validity (VALID), target format accuracy (FORMAT), sequence similarity (SIM), BLEU, CHRF, JSON parseability (JSON), operator validity (OP), and IDS output-level precision and recall, which are used to compute the F_1 -score. Let Ω_{F_i} denote the set of valid operators supported by the target rule format F_i (e.g., the operators permitted by a specific WAF platform). For N generated rules, the syntactic and operator validity rates are defined as:

$$\begin{aligned} V_{syn} &= \frac{1}{N} \sum_{i=1}^N \mathbf{1}[\text{validate}(\hat{y}_i, F_i) = 1], \\ V_{op} &= \frac{1}{N} \sum_{i=1}^N \mathbf{1}[\text{op}(\hat{y}_i) \subseteq \Omega_{F_i}]. \end{aligned} \quad (3)$$

²ApplicationGatewayWebApplicationFirewallPolicy

IDS output-level precision, recall, and F_1 are computed over predictions in the merged IDS class:

$$\begin{aligned} P &= \frac{TP}{TP + FP}, \quad R = \frac{TP}{TP + FN}, \\ F_1 &= \frac{2PR}{P + R}. \end{aligned} \quad (4)$$

The live deployment tier adds three provider-facing metrics: AWS live acceptance rate and actual WCU from `wafv2:CheckCapacity`, Cloudflare API acceptance rate from the Rulesets API or expression validator, and Azure API acceptance rate from ARM provisioning. All three metrics reflect real provider decisions, not offline approximations.

The *static deployability proxy rate* is the complementary offline metric, in which it is the fraction of generated rules that pass all local validators (JSON schema check, operator-set audit, and structural validity) without submitting to any live API. It serves as a low-latency pre-filter that identifies structurally malformed rules before incurring API round-trip cost. The gap between static proxy and the corresponding live acceptance rates reflects provider-side runtime constraints that offline validators cannot replicate.

4.2 Data and Threat-Pattern Coverage

Figure 4 summarizes what the specialized agents/adapters cover. The IDS agent emits Snort and Suricata signatures for web scans, suspicious HTTP methods and URIs, Telegram bot API activity, malware and trojan indicators, DNS and TLS artifacts, phishing infrastructure, credential abuse, exfiltration, C2, reconnaissance, and DoS-style categories. Further, Cloud WAF data coverage spans custom match rules, rate-based rules, managed-rule hooks, API-shield contexts, IP-access contexts, and block or log actions over headers, and user agents. Pattern diversity includes mixed-case, URL-decoded, double-encoded, and Base64-hinted variants, which lets the agents draft rules for newly observed payload variants once they are supplied by threat intelligence or analyst tickets.

Figure 4 shows a list of covered attacks in the generated rules, demonstrating that the trained LLMs cover diverse vulnerability families rather

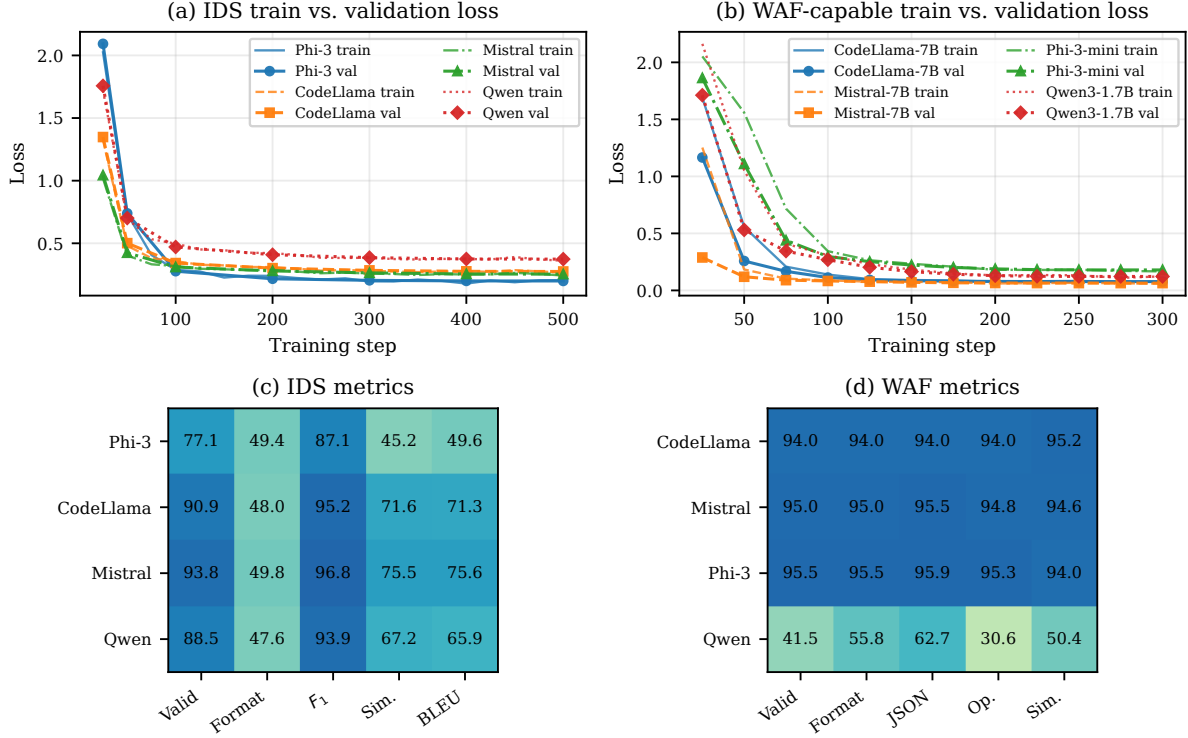


Fig. 3 Evaluation summary. The top row compares train and validation loss for IDS adapters (a) and WAF adapters (b), confirming convergence without overfitting. The bottom row reports IDS (c) and WAF (d) metric heatmaps summarizing validity, format fidelity, semantic similarity, and text quality across model families.

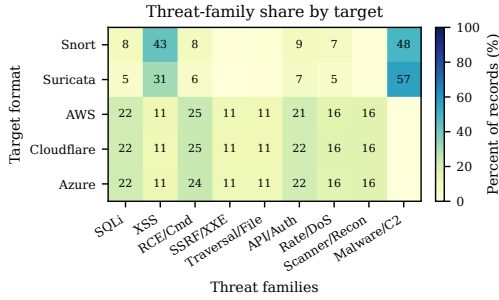


Fig. 4 Threat family coverage by target corpus. The figure shows that the agents see both classical IDS categories and cloud application attacks, supporting the coverage contribution of WAFGen.

than a narrow set of syntactic examples, enabling users to create more comprehensive and dynamic IDS rules.

The generated WAF rules by the trained models show emphasis on the **custom** and **Block** type rules. Figure 5 illustrates a comprehensive set of rule-type coverage by the trained WAF models, while not focusing on generating **Managed** rules since

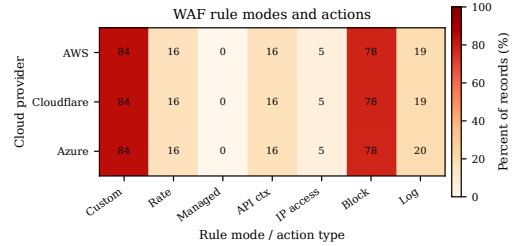


Fig. 5 WAF rule mode and action coverage across AWS, Cloudflare, and Azure. The figure shows the covered corpus by the WAF agents on rule types such as custom, rate, managed, API, IP, block, and log controls.

they produce more complex Cloudflare and AWS WAF structures, hence causing higher WCU costs (for the AWS WAF case).

4.3 Results

The training and evaluation results cover four model families evaluated under two main settings. The IDS setting evaluates 1,000 held-out Snort and

Table 3 Full held-out generation metrics. Values are percentages except BLEU and chrF. The JSON/ F_1 column reports IDS F_1 for IDS models and JSON parseability for WAF models. IDS P/R reports output-level precision and recall when applicable.

Setting	Model	N	Valid	Format	JSON/ F_1	Op.	Sim.	BLEU	chrF	IDS P/R
IDS	PHI-3-MINI	1000	77.1	49.4	87.1	77.1	45.2	49.6	59.3	100.0/77.1
	CODELLAMA-7B	1000	90.9	48.0	95.2	90.9	71.6	71.3	78.2	100.0/90.9
	MISTRAL-7B	1000	93.8	49.8	96.8	93.8	75.5	75.6	81.9	100.0/93.8
	QWEN3-1.7B	1000	88.5	47.6	93.9	88.5	67.2	65.9	73.5	100.0/88.5
WAF	PHI-3-MINI	1000	95.5	95.5	95.9	95.3	94.0	92.4	93.7	—
	CODELLAMA-7B	1000	94.0	94.0	94.0	94.0	95.2	92.7	95.1	—
	MISTRAL-7B	1000	97.6	98.0	98.0	97.2	97.7	96.2	97.6	—
	QWEN3-1.7B	1000	86.8	86.8	86.8	82.4	85.7	81.7	85.7	—

Suricata records using PHI-3-MINI, CODELLAMA-7B, MISTRAL-7B, and QWEN3-1.7B IDS adapters. The WAF live deployment setting evaluates 250 records (75 AWS, 100 Cloudflare, 75 Azure) using WAF adapters for the same four model families, with all three provider APIs called live via the mechanisms described in Section 3.6. Generation uses bounded output length, temperature 0.1, top-p 0.9, and repetition penalty 1.12. Figure 3 shows that the CODELLAMA-7B, MISTRAL-7B, and PHI-3-MINI adapters perform almost perfectly on all WAF metrics, while QWEN3-1.7B performing poorly. On the other hand, all adapters achieve decent to excellent performance on the IDS rule validity and F_1 -scores. Further investigation and possibly another enhanced training round is needed to improve the models’ performances on the rest of metrics for both rule families (IDS & WAF).

Table 3 reports the acquired trained-model results. Overall, the 7B adapters perform best across IDS quality and WAF text fidelity: MISTRAL-7B reaches the best IDS schema and operator validity at 93.8%, 81.9 chrF, and 96.8% IDS F_1 . MISTRAL-7B also dominates the performance in WAF rules generation on all metrics.

Among small models (3B or less), PHI-3-MINI remains competitive given its smaller size compared to MISTRAL-7B and CODELLAMA-7B, with 95.5% WAF validity and format accuracy, while scoring 95.3% for JSON parseability.

The Format accuracy metric values introduced in Table 3 (47–50% for IDS rules generators) shows a poor performance. Nevertheless, the format accuracy is influenced by the high similarities between Snort and Suricata rules structures and fields, leading the IDS models’ output to confuse each rule’s type for the other one, explaining the

near-random 47.6–49.8% rate. All models maintain high syntactic validity and operator validity, indicating that the Snort/Suricata structural differences do not prevent correct rule generation and deployment. For WAF, Format accuracy is high (86.8–98.0%) because providers have more distinct JSON schemas. Hence, leading the models to learn provider-specific field structures correctly.

Live Deployment Results. Table 4 reports the live provider acceptance rates across 250 WAF rules per model (75 AWS, 100 Cloudflare, 75 Azure, the imbalance is inherited from the original dataset) submitted to real cloud APIs. MISTRAL-7B achieves the highest overall results with 96% AWS acceptance, 76% for Cloudflare, and 90.67% for Azure. CODELLAMA-7B reaches 93.33% on AWS and 75% on Cloudflare. QWEN3-1.7B shows a decent AWS deployment acceptance levels reaching 72%, despite its compact size. Mean API round-trip latency ranges between 237 ms (PHI-3-MINI) to 257 ms (MISTRAL-7B), well within interactive tooling budgets.

AWS WCU Budget Control. AWS Web ACL Capacity Units are a published hard constraint on rule complexity and a direct billing lever. Every Web ACL has a default ceiling of 1,500 WCU (extendable at cost), and each rule statement type carries a fixed WCU unit cost documented in the AWS WAF developer guide [3]. A `ByteMatchStatement` costs 1 WCU, a `RegexMatchStatement` costs 3 WCU, and the popular `AWSMangedRulesCommonRuleSet` alone consumes 700 WCU—nearly half the default Web ACL budget—leaving limited headroom for additional custom rules. WAFGen integrates WCU measurement into the rule-generation loop by calling `wafv2:CheckCapacity` with the exact generated rule JSON before any WebACL is created.

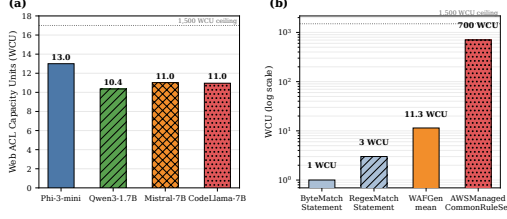


Fig. 6 WAFGen actual WCU per model (a, from `wafv2:CheckCapacity`) versus published AWS statement costs and the `AWSManagedRulesCommonRuleSet` reference (b). WAFGen rules average 10.4–13.0 WCU is far below the other Web ACL ceiling costs.

Table 4 Live deployment results. WCU is measured with `wafv2:CheckCapacity`; latency is the mean API round trip.

Model	<i>N</i>	AWS%	WCU	CF%	Azure%	Lat. (ms)
Phi-3-MINI	250	26.67	13.0	44.0	38.67	237
QWEN3-1.7B	250	72.0	10.4	66.0	54.67	243
MISTRAL-7B	250	96.0	11.0	76.0	90.67	257
CODELLAMA-7B	250	93.33	11.0	75.0	50.67	243

The service returns the integer WCU the rule would consume, giving the operator accurate cost information at rule-draft time rather than after deployment.

Figure 6 places the measured results in context: across all four model families, WAFGen-generated rules yield mean actual WCU values of 10.352 (QWEN3-1.7B), 10.957 (CODELLAMA-7B), 11.014 (MISTRAL-7B), and 13.0 (PHI-3-MINI), all well below the byte-match baseline and far below managed-ruleset costs. Critically, every model achieves a zero WCU delta versus the reference human-authored rules, confirming that fine-tuning does not inflate rule complexity relative to baselines. Security teams can therefore set a WCU threshold in their CI pipeline, rejecting over-budget rule drafts automatically before any cloud resource is touched, while preserving Web ACL headroom for additional rule families.

4.4 Provider Specialization vs. Multi-Platform Generalization

From this section onwards, our focus is directed towards only the WAF rules generation models. We distinguish three adapter families throughout this subsection, *Single-v1* adapters are trained exclusively on one provider’s rule corpus (4,250 records, 300 training steps); they represent the minimum-data, provider-focused baseline. *Single-v2* adapters

are data-equalized variants that has an augmented corpus size, three times the first version to reach the same 12,750-record training volume as the multi-platform adapters, and are trained to 500 steps. The equal dataset sizes for *Single-v2* and multi-platform model families ensures fair performance comparison, and would allow to witness the impact on adapters performance differences. *Multi* adapters are trained on all three providers’ corpora simultaneously (12,750 records, 500 steps). They are general-purpose WAF generation agents described in Section 4. The four models used in the experiments are abbreviated as **Phi3** (PHI-3-MINI, 3.8B), **Mistral** (MISTRAL-7B-Instruct), **CL7B** (CODELLAMA-7B-Instruct), and **Qwen3** (QWEN3-1.7B-Base).

Reported metrics. *Valid%* is the offline syntactic and schema validity rate, whether the generated text parses and passes all provider-specific field constraints without calling any live API. *BLEU* and *chrF* are token-level *n*-gram text similarity scores between generated and reference rules. Higher values indicate closer structural and lexical alignment with human-authored examples. *Live%* is the real cloud API acceptance rate: the fraction of generated rules accepted by the live AWS `wafv2:CheckCapacity` endpoint, the Cloudflare Rulesets API, or the Azure ARM provisioning service, respectively. *Mean WCU* (AWS only) is the mean Web ACL Capacity Unit cost returned by `wafv2:CheckCapacity`. Lower values indicate cheaper, more targeted rules that preserve Web ACL budget headroom.

To quantify the trade-off between single-provider focus and cross-provider flexibility, we train an additional set of 12 QLoRA adapters—one per model family per cloud provider—under identical hyperparameters but with training data restricted to a single provider’s rule corpus (*single-v1* adapters). A second set of 12 *single-v2* adapters is trained with the same corpus augmented to match the multi-platform data volume, isolating the effect of data *diversity* from data *quantity*. These are compared against the WAF-capable adapters trained on all three providers simultaneously (*multi* adapters). Each adapter is evaluated on each provider’s 250-record test split, and live acceptance rates are measured through the same real API calls described in Section 3.6.

Validity overview. Figures 7 and 8 present the offline validity (checking locally the syntactic correctness of generated rules without deployment) breakdown across all 36 adapter×provider combinations. In Figure 7, Mistral multi achieves perfect validity (100%) on all three providers. CodeLlama multi reaches 98.8% on Cloudflare and 94.4% on Azure, compared to 79.2% and 69.6% for its single-v1 counterpart. PHI-3-MINI multi reaches 97.2%/93.2%/97.6% on AWS/CF/Azure versus 14.4%/69.6%/75.0% for single-v1 and 2.4%/94.0%/87.2% for single-v2. The companion Figure 8 shows BLEU and chrF scores consistently tracking validity trends: adapters with low validity produce fragmented or mis-formatted outputs with degraded text scores, while multi-platform adapters maintain high fidelity across all three dimensions simultaneously.

Performance delta analysis. Figure 9 quantifies the per-metric improvement of multi-platform adapters over the single-v1 baseline ($\Delta = \text{Multi} - \text{Single-v1}$). Positive deltas confirm that multi-platform training is broadly beneficial: on average, validity gains of +30 to +70 percentage points are observed for PHI-3-MINI and CodeLlama across AWS and Azure, with BLEU improvements of +10 to +40 points. For Cloudflare, Mistral single-v1 already achieves 100% validity, leaving no room for improvement—a ceiling effect visible as a near-zero delta bar. Notably, data-augmented single-v2 adapters do not consistently close the multi-platform gap, indicating that corpus *diversity*—not corpus *size*—is the dominant factor driving multi-platform superiority.

Table 5 Mean AWS WCU by model and adapter type for rules accepted by the live `wafv2:CheckCapacity` API. N/A indicates no valid AWS rules were produced.

Model	Single-v1	Single-v2	Multi
PHI-3-MINI	12.38	9.86	21.69
MISTRAL-7B	21.67	22.58	21.78
CODELLAMA-7B	25.00	20.63	21.05
QWEN3-1.7B	19.06	N/A	25.00

Multi-dimensional radar comparison.

Figure 10 presents one radar chart per model family, arranged in a horizontal row. Each chart has nine spokes—one for every (provider, training condition) pair (AWS, CF, and Azure crossed with Single-v1, Single-v2, and Multi)—where the radial value on each spoke is the syntactic *Valid%* for

that combination. A single filled polygon traces the model’s validity profile across all nine combinations, the dashed grey ring marks the 90% reference threshold. MISTRAL-7B fills nearly the entire available area on all nine spokes, confirming uniformly high validity regardless of provider or training condition. CODELLAMA-7B and PHI-3-MINI show a clear asymmetry, where the Multi-condition spokes (rightmost group per provider) are consistently longer than their Single-v1 counterparts, confirming the benefit of multi-platform training. QWEN3-1.7B exposes the most irregular shape, its Azure spokes (all three conditions) reach full radius at 99%, while its AWS and Cloudflare spokes collapse to near zero, revealing the limited capacity of inter-format interference discussed above.

Heatmap summary. Figure 11 presents the complete live acceptance surface as three provider heatmaps (rows: model families, columns: target cloud platforms). The AWS heatmap (a) highlights MISTRAL-7B multi at 100% followed by PHI-3-MINI multi at 97.2%, which achieves no less than 91.0% for all cloud platforms, given its small size compared to the 7B models. The Cloudflare heatmap (b) shows the most consistent multi-platform advantage, MISTRAL-7B (100%), CODELLAMA-7B (98.8%), and PHI-3-MINI (93.2%) all surpass their single-platform counterparts. The Azure heatmap (c) exposes the QWEN3-1.7B anomaly most clearly. Single-v1 and single-v2 both at 99.2% while multi at 0.8%, and demonstrates Mistral’s balanced 100% sweep across all three adapter conditions. Overall, the multi-platform outperformed single-platform adapters in the three cloud service providers, where the single-v1 family showed the weakest performance for AWS, and QWEN3-1.7B performing poorly almost under every setting (except under Azure’s platform).

AWS WCU under specialization conditions. `wafv2:CheckCapacity` measurements illustrated in Table 5, show that multi-platform training increase slightly rule complexity, hence the average WCU cost (when comparing Figure 6 and Table 5). This increase is linked to the slight improvement in models performance due to the augmentation of training dataset size, which means generating more correct and valid rules would logically increase the WCU cost per rule.

MISTRAL-7B (multi platform adapter) achieves 21.78 WCU versus 21.67 for single-v1 (difference

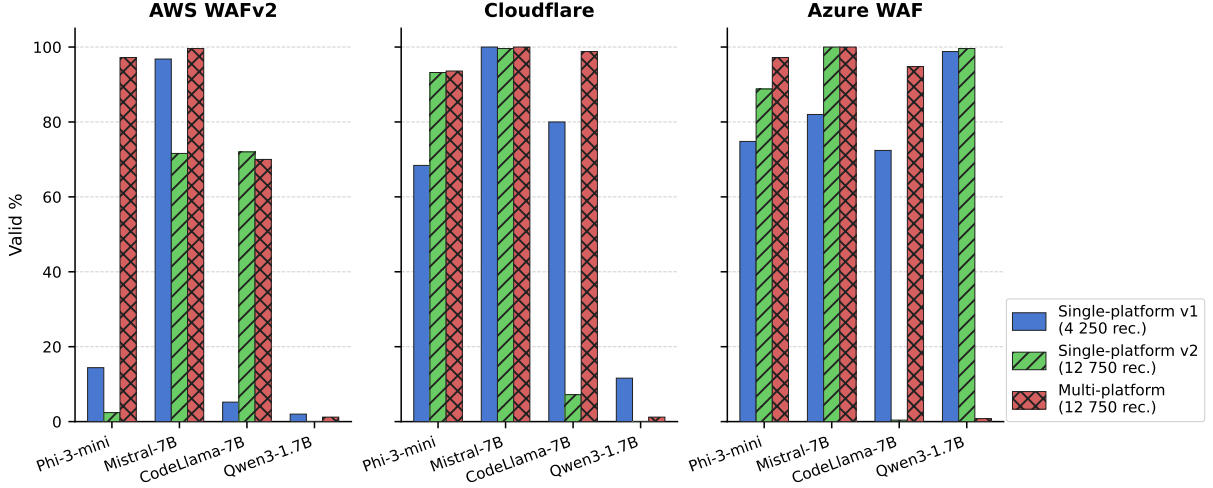


Fig. 7 Syntactic validity (%) across all adapter types (Single-v1, Single-v2, Multi) and provider targets. Each grouped-bar cluster represents one model family. Bar colour encodes the adapter type. Multi-platform adapters (green bars) consistently reach or approach 100% on every provider, while single-platform adapters can collapse to near zero even on the provider they were trained for, particularly when data volume is low (Single-v1).

<1%), and CODELLAMA-7B single-v1 reports 25.0 WCU (one degenerate accepted rule), while single-v2 and multi converge to 20.63 and 21.05 respectively. PHI-3-MINI is an outlier at 12.38/9.86 WCU for single-v1/single-v2 adapters, lower than expected because the small number of valid rules it produces are simple byte-match constructs. All values remain well below the 700 WCU cost of `AWSManagedRulesCommonRuleSet` and the 1,500 WCU WebACL ceiling, confirming that adapter training condition does not affect budget efficiency.

Multi-platform training vs. single-platform specialization. The results above consistently favour multi-platform adapters for three of the four model families, and the single-v2 experiment confirms that this advantage is not merely a function of training-set size. We attribute the gap to a *cross-model regularization* effect. When a model jointly learns AWS WAFv2 JSON statements, Cloudflare expression-language policies, and Azure ARM condition objects in the same training run, it is forced to internalize the structural and functional *differences* between provider schemas rather than the surface-level token distributions of a single format. This shared representation implicitly suppresses hallucinations of invalid field names, unsupported operators, and schema-mixing artifacts, which represent the dominant failure instances for single-platform adapters even when tested on the same provider they were

trained for. Conversely, single-platform adapters trained on a narrow corpus can overfit to the lexical patterns of one provider, losing the ability to distinguish key structural primitives. The QWEN3-1.7B exception reinforces this interpretation from the opposite side. At 1.7B parameters, the model has insufficient capacity to encode three distinct provider grammars simultaneously, causing catastrophic interference and near-zero generation quality on AWS and Cloudflare in the multi-platform setting.

The results confirm that multi-platform training is the right default for operators who need a single deployable agent covering all three cloud providers. Single-platform specialization is preferable only when the deployment target is fixed, model capacity is limited, and the provider’s rule language is sufficiently distinct to reward schema-focused training (e.g., QWEN3-1.7B single-v2 on Azure).

Together, Figures 3–11 and Table 4 support the paper’s main contribution: target-aware adapters converge cleanly, generate syntactically valid rules with high threat coverage, and, for the 7B model families, produce WAF rules that are accepted by real cloud provider APIs at rates exceeding 90% on AWS and Azure, with low WCU complexity that helps keeping cloud security budgets under operator control.

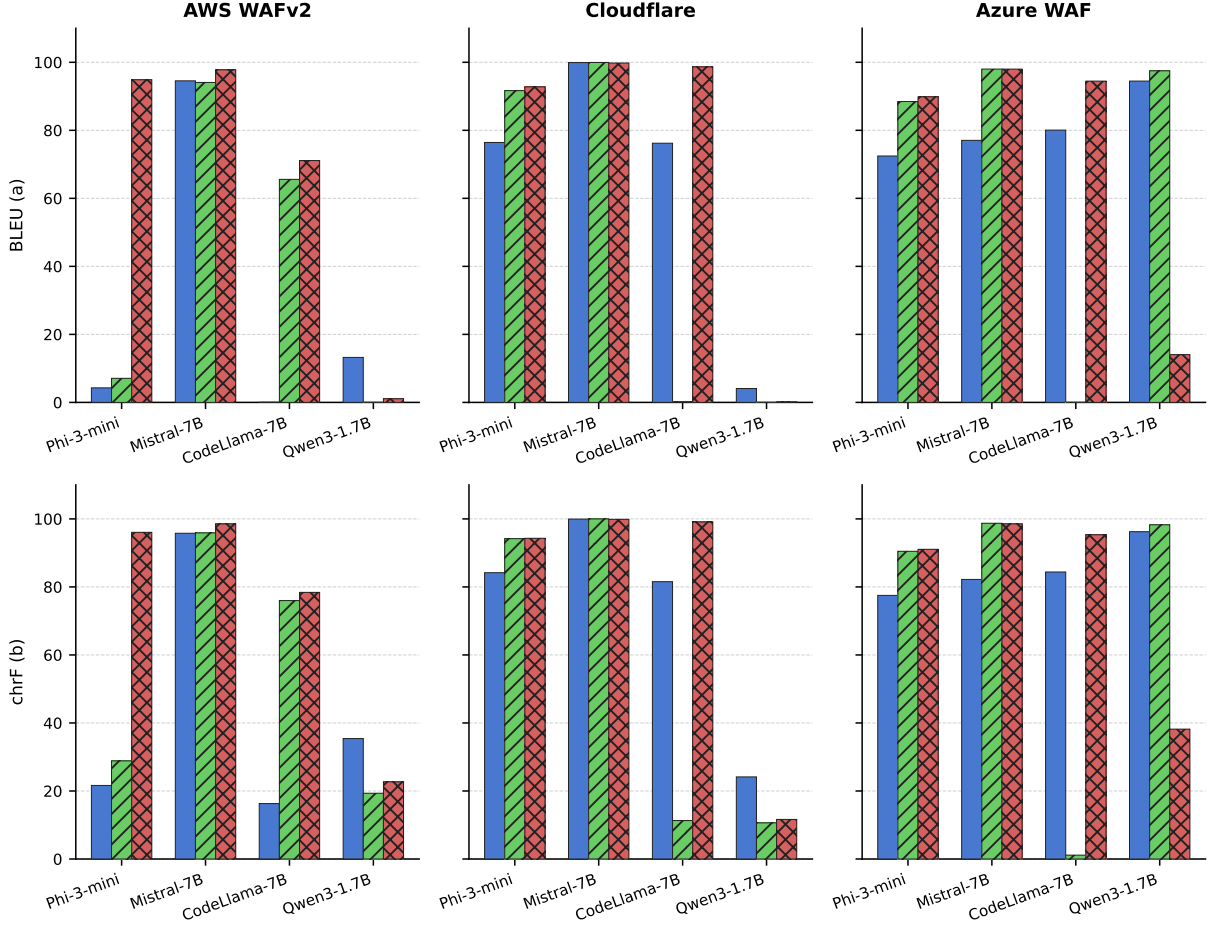


Fig. 8 BLEU (a) and chrF (b) text similarity scores for generated WAF rules, broken down by provider (columns) and adapter type (bar colour). Multi-platform adapters dominate on AWS and Cloudflare; the advantage is less pronounced for Azure, where Mistral Single-v2 and Qwen3 Single-v1 achieve competitive scores at the cost of cross-provider generalization.

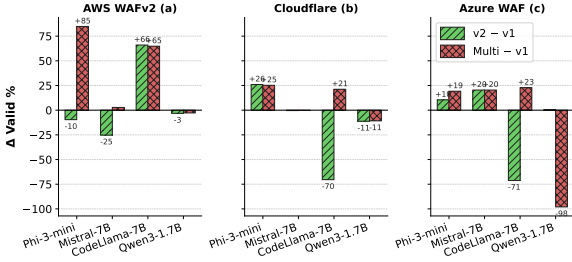


Fig. 9 Performance difference between Multi and Single-v1 adapters ($\Delta = \text{Multi} - \text{Single-v1}$) by metric and provider. Positive bars indicate gains from multi-platform training, whereas negative bars favor single-platform training. The largest improvements are observed for PHI-3-MINI and CodeLlama on AWS, suggesting that the AWS JSON schema benefits most from cross-provider training.

5 Conclusion, Limitations, and Future Work

This paper presented WAFGen, an LLM framework for generating Snort, Suricata, AWS WAF, Cloudflare WAF, and Azure WAF rules (with emphasis on the three cloud service providers). The system extends our previous work’s [7, 8] models, and trains multiple adapters capable of generating detection rules for IDS, and multi-platform WAF rules for AWS, CF, and Azure.

Our extensive experiment design covered several evaluation aspects, leading us to refute our initial hypothesis regarding *Multi-platform* vs. *Single-platform* setups viability: (1) models performance on generating IDS and WAF rules achieving

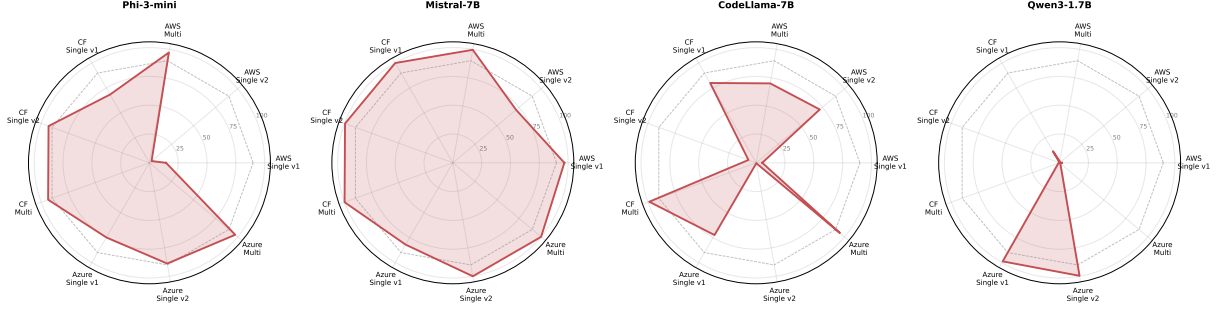


Fig. 10 Valid% radar charts for each model family (one panel per model, arranged in a single row). Each chart has nine spokes representing all provider $\times$ training-condition combinations: AWS, Cloudflare (CF), and Azure, each crossed with Single-v1, Single-v2, and Multi. The radial value on each spoke is the syntactic validity rate (%), the dashed grey ring marks 90%. A larger, rounder polygon indicates high and consistent validity across all conditions. MISTRAL-7B nearly fills the full radar on all nine spokes, where QWEN3-1.7B shows a strongly asymmetric profile with near-perfect Azure but near-zero AWS and CF validity in all conditions.

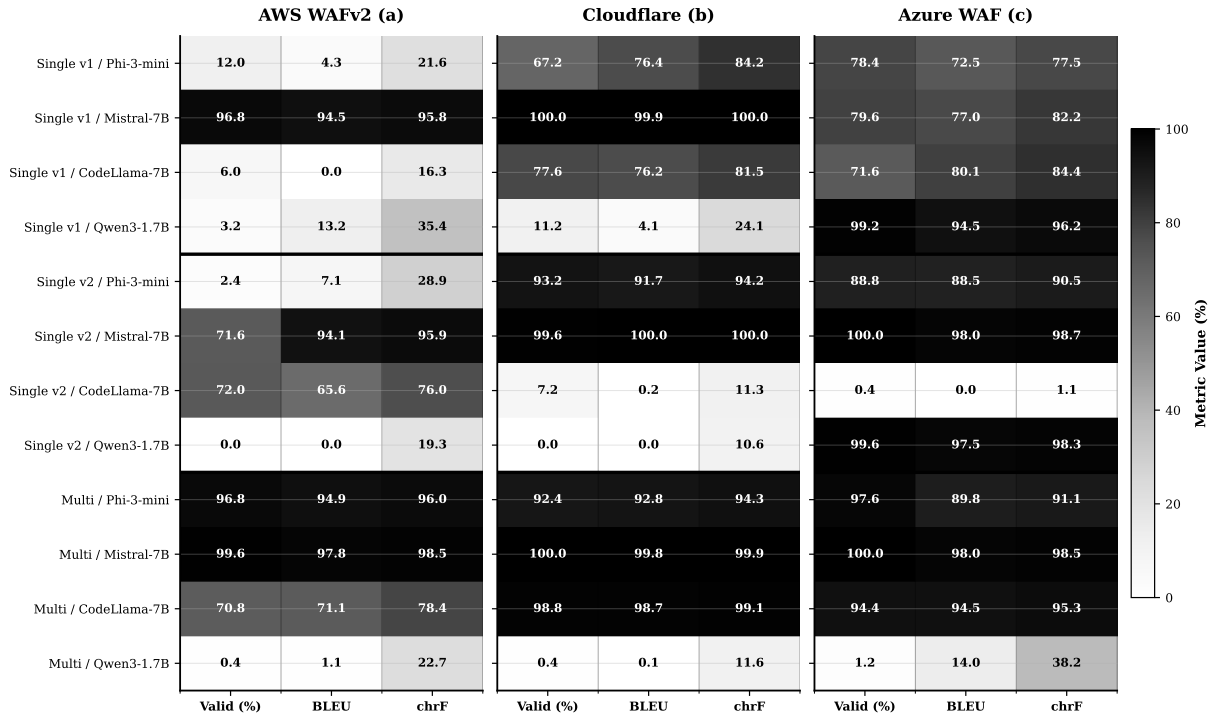


Fig. 11 Live API acceptance rate heatmaps for AWS (a), Cloudflare (b), and Azure (c). Rows represent model families. Columns represent adapter training conditions (Single-v1, Single-v2, Multi). Cell values are live acceptance percentages, where darker cells indicate higher live deployment/acceptance rates. The Multi column is consistently dark for MISTRAL-7B and CODELLAMA-7B across all three providers, while QWEN3-1.7B shows a unique inversion between Azure (dark single, bright multi) and AWS/CF (bright single and multi).

above 93% scores on operation and syntactic validity (for MISTRAL-7B). (2) studying the effect of collectively or separately training the models on the WAF data corpus, concluding that multi-platform adapters training reduces rule fields hallucination (cross-model regularization) while achieving over

90% live API deployability on all cloud service platforms. (3) Achieving cost-effective generated AWS WAF rules (21.67 WCU by MISTRAL-7B), leaving headroom relative to the 1,500 WCU Web ACL ceiling and the 700 WCU cost of the AWS managed

common ruleset. This feature would assist AWS services’ users assessing and saving their WCU budget before deploying any WAF rules. (4) Small models (PHI-3-MINI and QWEN3-1.7B), outperformed by MISTRAL-7B and CODELLAMA-7B models, can achieve decent IDS and WAF rule generation performance only for a few cloud platforms. Use cases of the trained models can vary from a user to another according to their deployment requirement, computing budget, and platform usage.

Future work will add traffic replay testbeds, cost-aware decoding that optimizes for WCU targets, retrieval from live threat intelligence feeds, and analyst feedback learning. We also plan to release WAFGen as an open-source plugin for major cloud security workflows and to investigate whether further multi-provider improves cross-cloud portability.

Declarations

Funding

Anes Abdennebi and Nadjia Kara received funding from the Ministère de la Défense Nationale, Canada, for this work.

Competing interests

The authors declare that they have no competing interests.

Data availability

The code, used datasets, and generated datasets are available at <https://github.com/RoronoaZ/WAFGen>.

Code availability

The implementation used in this study is available at <https://github.com/RoronoaZ/WAFGen>.

Author contributions

Anes Abdennebi: Conceptualization, Methodology, Software, Validation, and Writing – original draft. Nadjia Kara: Supervision and Writing – review and editing. Hakima Ould Slimane: Supervision and Writing – review and editing. All authors read and approved the final manuscript.

References

- [1] OWASP Foundation: OWASP Top Ten 2021. <https://owasp.org/www-project-top-ten/>. Accessed: May 2025 (2021)
- [2] Verizon Business: 2024 Data Breach Investigations Report. Technical report, Verizon Communications Inc. (2024). <https://www.verizon.com/business/resources/reports/dbir/>
- [3] Amazon Web Services: AWS WAF Developer Guide. <https://docs.aws.amazon.com/waf/>. Accessed: May 2026 (2025)
- [4] Microsoft: Azure Web Application Firewall Documentation. <https://learn.microsoft.com/azure/web-application-firewall/>. Accessed: May 2026 (2025)
- [5] Cloudflare: Ruleset Engine and WAF Documentation. <https://developers.cloudflare.com/ruleset-engine/>. Accessed: May 2026 (2025)
- [6] Tracy, M., Baker, W., Jansen, W.: Guidelines on Securing Public Web Servers. Technical Report SP 800-44 Version 2, National Institute of Standards and Technology (2007). <https://csrc.nist.gov/publications/detail/sp/800-44/ver-2/final>
- [7] Abdennebi, A., Kara, N., Lahlou, L., Ould-Slimane, H.: LI-NIDS: LLM-based intelligent NIDS rules generation for cybersecurity applications. In: 2025 16th International Conference on Network of the Future (NoF). IEEE, ??? (2025). <https://doi.org/10.1109/NOF66640.2025.11223303>
- [8] Anes, A., Nadjia, K., Laaziz, L., Hakima, O.-S.: LanG – A Governance-Aware Agentic AI Platform for Unified Security Operations (2026). <https://arxiv.org/abs/2604.05440>
- [9] Gungor, O., Sood, R., Wang, H., Rosing, T.: AQUA-LLM: Evaluating accuracy, quantization, and adversarial robustness trade-offs in LLMs for cybersecurity question answering. arXiv preprint arXiv:2509.13514 (2025)
- [10] Bhatt, M., Chennabasappa, S., Li, Y., Nikolaidis, C., Song, D., Wan, S., Ahmad, F.,

- Aschermann, C., Chen, Y., Kapil, D., Molnar, D., Whitman, S., Saxe, J.: CyberSecEval 2: A wide-ranging cybersecurity evaluation suite for large language models. arXiv preprint arXiv:2404.13161 (2024)
- [11] Wan, S., Nikolaidis, C., Song, D., Molnar, D., Crnkovich, J., Grace, J., Bhatt, M., Chennabasappa, S., Whitman, S., Ding, S., Ionescu, V., Li, Y., Saxe, J.: CyberSecEval 3: Advancing the evaluation of cybersecurity risks and capabilities in large language models. arXiv preprint arXiv:2408.01605 (2024)
- [12] Singh, R., et al.: LLMs in the SOC: An empirical study of human-AI collaboration in security operations centres. arXiv preprint arXiv:2508.18947 (2025)
- [13] Song, C., et al.: Audit-LLM: Multi-agent collaboration for log-based insider threat detection. arXiv preprint (2024). Cited in arXiv:2603.18196 and arXiv:2508.18947
- [14] Sun, Y., Wu, D., Xue, Y., Liu, H., Ma, W., Zhang, L., Liu, Y., Li, Y.: Llm4vuln: A unified evaluation framework for decoupling and enhancing llms' vulnerability reasoning. arXiv preprint arXiv:2401.16185 (2024)
- [15] Li, Z., Dutta, S., Naik, M.: IRIS: LLM-assisted static analysis for detecting security vulnerabilities. In: The Thirteenth International Conference on Learning Representations (ICLR) (2025)
- [16] Marais, B., Quertier, T., Barrue, G.: Semantic preprocessing for llm-based malware analysis. arXiv preprint arXiv:2506.12113 (2025)
- [17] Fang, R., Bindu, R., Gupta, A., Zhan, Q., Kang, D.: LLM agents can autonomously exploit one-day vulnerabilities. arXiv preprint arXiv:2404.08144 (2024)
- [18] Alhuzali, A.: Llm-powered threat intelligence: a retrieval-augmented generation approach for cyber attack investigation. PeerJ Computer Science **11**, 3371 (2025)
- [19] Ullah, S., Balasubramanian, P., Guo, W., Burnett, A., Pearce, H., Kruegel, C., Vigna, G., Stringhini, G.: From cve entries to verifiable exploits: An automated multi-agent framework for reproducing cves. arXiv preprint arXiv:2509.01835 (2025)
- [20] Lotfi, A., Katsis, C., Bertino, E.: Automated vulnerability validation and verification: A large language model approach. arXiv preprint arXiv:2509.24037 (2025)
- [21] Ghosh, S.: Network traffic analysis based on cybersecurity intrusion detection through an effective automated separate guided attention federated graph neural network. Applied Soft Computing **169**, 112603 (2025)
- [22] Appiahene, P., Berchie, S.O., Botchway, E., Ayitey, M.J., Dawson, J.K., Mettle, H.N.-A., Afrifa, S.: Network intrusion detection using a hybrid graph-based convolutional network and transformer architecture. PloS one **21**(1), 0340997 (2026)
- [23] Guerra, L., Chapuis, T., Duc, G., Mozharovskiy, P., Nguyen, V.-T.: Self-supervised learning of graph representations for network intrusion detection. Advances in Neural Information Processing Systems **38**, 109471–109501 (2026)
- [24] Zixuan, W., Cheng, M., Yuhua, X., Zeyi, L., Zhixin, S., Pan, W.: Trusted encrypted traffic intrusion detection method based on federated learning and autoencoder. China Communications **21**(8), 211–235 (2024)
- [25] Cisco: Snort 3 Rule Writing Guide. <https://docs.snort.org/>. Accessed: May 2026 (2025)
- [26] Open Information Security Foundation: Suricata User Guide. <https://docs.suricata.io/>. Accessed: May 2026 (2025)
- [27] OWASP Foundation: OWASP Core Rule Set Documentation. <https://coreruleset.org/docs/>. Accessed: May 2026 (2025)
- [28] Scano, C., Floris, G., Montaruli, B., Demetrio, L., Valenza, A., Compagna, L., Ariu, D., Piras, L., Balzarotti, D., Biggio, B.: Modsec-learn: Boosting modsecurity with machine learning. In: International Symposium on Distributed

Computing and Artificial Intelligence, pp. 23–33 (2024). Springer

- [29] Lian, W., Zhang, C., Zhang, H., Jia, B., Liu, B.: Rulemaster+: Llm-based automated rule generation framework for intrusion detection systems. *Chinese Journal of Electronics* **34**(5), 1402–1415 (2025)
- [30] Papoutsis, A., Dimitriadis, A., Koritsas, I., Kavallieros, D., Tsikrika, T., Vrochidis, S., Kompatsiaris, I.: Rulexploit: A framework for generating suricata rules from exploits using generative ai. In: 2025 IEEE International Conference on Cyber Security and Resilience (CSR), pp. 267–274 (2025). IEEE
- [31] Balasubramanian, P., Ali, T., Salmani, M., KhoshKholgh, D., Kostakos, P.: Hex2sign: automatic ids signature generation from hexadecimal data using llms. In: 2024 IEEE International Conference on Big Data (BigData), pp. 4524–4532 (2024). IEEE
- [32] Hu, X., Chen, H., Bao, H., Wang, W., Liu, F., Zhou, G., Yin, P.: A llm-based agent for the automatic generation and generalization of ids rules. In: 2024 IEEE 23rd International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), pp. 1875–1880 (2024). IEEE
- [33] Mitra, S., Bazarov, A., Duclos, M., Mittal, S., Piplai, A., Rahman, M.R., Ziegler, E., Rahimi, S.: Falcon: Autonomous cyber threat intelligence mining with llms for ids rule generation. arXiv preprint arXiv:2508.18684 (2025)
- [34] Li, J., Chai, Y., Du, L., Duan, C., Yan, H., Gu, Z.: Gridai: Generating and repairing intrusion detection rules via collaboration among multiple llm-based agents. arXiv preprint arXiv:2510.13257 (2025)
- [35] Wang, H., Xu, M., Guo, Y., Han, W., Lim, H.W., Dong, J.S.: Rulepilot: An llm-powered agent for security rule generation. arXiv preprint arXiv:2511.12224 (2025)
- [36] Toprani, D., Madiseti, V.K.: Llm agentic workflow for automated vulnerability detection and remediation in infrastructure-as-code. IEEE Access (2025)
- [37] Abdin, M., Jacobs, S.A., Awan, A.A., Aneja, J., Awadallah, A., Bach, N., Bahree, A., Bakhtiari, A., Behl, H., Benhaim, A., et al.: Phi-3 technical report: A highly capable language model locally on your phone. arXiv preprint arXiv:2404.14219 (2024)
- [38] Yang, A., Li, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Gao, C., Huang, C., Lv, C., et al.: Qwen3 technical report. arXiv preprint arXiv:2505.09388 (2025)
- [39] Roziere, B., Gehring, J., Gloeckle, F., Sootla, S., Gat, I., Tan, X.E., Adi, Y., Liu, J., Sauvestre, R., Remez, T., et al.: Code llama: Open foundation models for code. arXiv preprint arXiv:2308.12950 (2023)
- [40] Jiang, A.Q., Sablayrolles, A., Roux, A., Mensch, A., Savary, B., Bamford, C., Chaplot, D.S., Casas, D.d.l., Hanna, E.B., Bressand, F., et al.: Mixtral of experts. arXiv preprint arXiv:2401.04088 (2024)