www.revistaft.com.br

Revista FT | ISSN 1678-0817 | v. 30, n. 158, 2026 | DOI: 10.69849/e27xgm50

Detecção de ameaças em logs de segurança: comparação entre abordagem baseada em regras e análise com LLM

Threat detection in security logs: a comparison between a rules-based approach and analysis with LLM

Flávia Thereza da Fonceca[1]

RESUMO

Este artigo tem como objetivo comparar, de forma sistemática, duas formas de detectar e classificar payloads maliciosos em parâmetros HTTP: a abordagem clássica, baseada em expressões regulares, e o uso de modelos de linguagem de grande escala (LLMs). Este trabalho utiliza o dataset HttpParamsDataset, que reúne 31.067 registros divididos em cinco classes: tráfego normal, injeção de SQL, Cross-Site Scripting, travessia de diretório e injeção de comandos. Foram avaliados sete modelos de dois provedores — Anthropic (Claude Haiku 4.5, Sonnet 4.6 e Opus 4.6) e OpenAI (GPT-4o-mini, GPT-4.1-mini, GPT-4.1 e GPT-5.4) — em duas modalidades: análise textual pura e geração dinâmica de scripts regex. A metodologia emprega 30 sub amostragens de 500 registros cada, com testes estatísticos pareados para validar os resultados. Os resultados mostram que a análise textual via LLM supera a abordagem baseada em regras estáticas na maior parte dos experimentos: o Claude Haiku 4.5 atingiu Macro-F1 de 0,967, contra 0,867 do motor de regras ($p < 0,0001$), a um custo de US$ 1,96 para as 30 sub amostragens. Quando os LLMs foram instruídos a gerar scripts regex, o desempenho caiu abaixo do motor escrito à mão em todos os modelos testados. O custo total dos 15 experimentos foi de US$ 35,81, com tempo computacional agregado de 9,8 horas. Como os experimentos são estatisticamente independentes, foram

[1] Graduanda em Tecnologia em Gestão da Tecnologia da Informação — Faculdade de Tecnologia de Jundiaí Deputado Ary Fossen. E-mail: flafonceca@gmail.com

executados em paralelo (4 processos concorrentes para a Anthropic e 2 para a OpenAI, com ambos os provedores rodando em paralelo entre si), reduzindo o tempo de parede efetivo para aproximadamente 3,5 horas. Os achados sugerem que, para classificação de payloads, o uso direto de LLMs como classificadores textuais oferece melhor custo-benefício que abordagens baseadas em regras estáticas ou em geração automática de regex, indicando que sistemas híbridos — com regras regex como primeira camada e LLMs para casos ambíguos — são uma direção promissora para ambientes de produção.

Palavras-chave: segurança web; detecção de ameaças; LLM; expressões regulares; classificação de payloads; inteligência artificial; cibersegurança.

ABSTRACT

This paper aims to systematically compare two approaches for detecting and classifying malicious payloads in HTTP parameters: the classical rule-based approach using regular expressions, and the use of large language models (LLMs). This study used the HttpParamsDataset, which contains 31,067 records divided into five classes: normal traffic, SQL injection, Cross-Site Scripting, path traversal, and command injection. Seven models from two providers were tested — Anthropic (Claude Haiku 4.5, Sonnet 4.6, and Opus 4.6) and OpenAI (GPT-4o-mini, GPT-4.1-mini, GPT-4.1, and GPT-5.4) — in two modalities: pure textual analysis and dynamic regex script generation. The baseline was a static engine with 40 regex rules inspired by the OWASP ModSecurity Core Rule Set. Experiments were run over 30 sub-samples of 500 records each, with paired statistical tests for validation. Results show that LLM textual analysis outperforms the rule-based approach in almost every scenario: Claude Haiku 4.5 reached Macro-F1 of 0.967 against 0.867 for the rule engine ($p < 0.0001$), at a cost of US$ 1.96 for the 30 sub-samples. When LLMs were asked to generate regex scripts, performance dropped below the hand-written engine across all tested models. Total cost of the 15 experiments was US$ 35.81, with 9.8 hours of aggregate computation time. Since the experiments are statistically independent, they were run in parallel (4 concurrent processes for Anthropic and 2 for OpenAI, with both providers running in parallel to each other), reducing the effective wall-clock time to approximately 3.5 hours. The findings suggest that, for payload classification, the direct use of LLMs as textual classifiers offers better cost-effectiveness than approaches based on static rules or automatic regex generation,

indicating that hybrid systems — with regex rules as a first filtering layer and LLMs for ambiguous cases — represent a promising direction for production environments.

Keywords: web security; threat detection; LLM; regular expressions; payload classification; artificial intelligence; cybersecurity.

1. INTRODUÇÃO

A segurança de aplicações web tem ganhado importância crescente no cenário tecnológico atual. Ataques como injeção de SQL (SQLi), Cross-Site Scripting (XSS), injeção de comandos do sistema operacional (CMDi) e travessia de diretório (path traversal) são ameaças recorrentes em aplicações web e podem causar desde vazamentos de dados até o comprometimento total de servidores (OWASP FOUNDATION, 2021a, 2021b).

A defesa tradicional contra essas ameaças passa, em grande parte, por sistemas baseados em regras, como os Web Application Firewalls (WAFs), que usam expressões regulares para reconhecer padrões maliciosos no tráfego de entrada (RISTIC, 2017; FREDJ et al., 2021). Essa estratégia funciona bem para ataques conhecidos, mas apresenta limitações: tem dificuldade com variantes ofuscadas, exige atualização constante das regras e apresenta alta taxa de falsos negativos em categorias menos frequentes (MEHTA et al., 2023; CRESPO-MARTÍNEZ et al., 2023).

Nos últimos anos, os Modelos de Linguagem de Grande Escala (Large Language Models — LLMs), baseados na arquitetura Transformer (VASWANI et al., 2017), surgiram como uma alternativa promissora. Trabalhos recentes mostram que LLMs como o Claude (Anthropic) e o GPT (OpenAI) têm bons resultados em tarefas de cibersegurança, incluindo detecção de vulnerabilidades em código e análise de ameaças (YAO et al., 2024; ZHOU et al., 2025).

Apesar do crescente interesse em aplicações de LLMs para cibersegurança, a literatura ainda carece de estudos que comparem sistematicamente essa abordagem com os métodos baseados em regras, considerando múltiplos modelos de diferentes provedores, diferentes modalidades de uso e o custo real de operação. A maior parte dos trabalhos foca em um único modelo ou compara LLMs entre si, sem um baseline regex robusto. Diante desse cenário, este

trabalho busca responder: LLMs podem superar a detecção baseada em regras estáticas na classificação de payloads HTTP maliciosos? Em que configurações e a que custo? Para responder a essas perguntas, o objetivo deste trabalho é comparar, de forma sistemática, uma abordagem tradicional baseada em regex com vários modelos de LLM na classificação de payloads HTTP. São avaliados sete modelos de dois provedores (Anthropic e OpenAI) em duas modalidades: análise textual direta e geração dinâmica de scripts de classificação. A comparação leva em conta desempenho, custo financeiro e tempo de execução.

2. FUNDAMENTAÇÃO TEÓRICA

2.1 Ameaças em Aplicações Web

Os tipos de vulnerabilidades em aplicações web são catalogados por iniciativas como a OWASP (Open Web Application Security Project) e o Common Weakness Enumeration (CWE) (MITRE CORPORATION, 2024). Entre as mais prevalentes, destacam-se quatro categorias: (a) Injeção de SQL (SQLi), classificada pela OWASP na categoria A03:2021 – Injection, que acontece quando dados enviados pelo usuário não são validados ou sanitizados, permitindo que comandos SQL maliciosos cheguem diretamente ao banco de dados (OWASP FOUNDATION, 2021b; HALFOND; VIEGAS; ORSO, 2006); (b) Cross-Site Scripting (XSS), que também pertence à A03:2021 e é identificada como CWE-79, caracterizada pela injeção de scripts em páginas visualizadas por outros usuários (OWASP FOUNDATION, 2021b; HYDARA et al., 2015); (c) Injeção de Comandos (CMDi), catalogada como CWE-78, que ocorre quando entradas não sanitizadas chegam a um interpretador do sistema operacional e permitem execução arbitrária de comandos no servidor (OWASP FOUNDATION, 2021b); e (d) Travessia de Diretório (Path Traversal), enquadrada na categoria A01:2021 – Broken Access Control (CWE-22), que permite manipular caminhos de arquivo para navegar fora do diretório raiz da aplicação e acessar arquivos sensíveis do sistema (OWASP FOUNDATION, 2021a).

2.2 Detecção Baseada em Regras (Regex)

Uma das formas mais tradicionais e difundidas de detectar ameaças em payloads é usar expressões regulares para descrever padrões conhecidos de ataque. Essa abordagem tem vantagens claras: execução previsível, latência sub-milissegundo, ausência de custos operacionais e facilidade de auditoria, já que cada regra pode ser lida e entendida. As limitações aparecem diante de cenários fora do escopo dos ataques conhecidos: variantes

ofuscadas podem escapar, novas categorias exigem atualização manual das regras e classes com sintaxe muito variada tendem a gerar falsos negativos (APPELT et al., 2018).

2.3 Modelos de Linguagem de Grande Escala (LLMs)

Os LLMs são modelos de aprendizado profundo baseados na arquitetura Transformer (VASWANI et al., 2017), treinados em grandes volumes de dados textuais e capazes de compreender e gerar tanto linguagem natural quanto código. Na prática, são consumidos via API e cobram por quantidade de tokens processados (um token corresponde, em média, a cerca de 4 caracteres) (OPENAI, 2026). A cobrança é separada em tokens de entrada (o prompt enviado) e tokens de saída (a resposta gerada pelo modelo). A Tabela 1 apresenta os modelos avaliados neste estudo e suas respectivas faixas de preço.

Tabela 1 — Modelos avaliados e precificação (USD por milhão de tokens)

Provedor	Modelo	Faixa	Entrada	Saída
Anthropic	Claude Haiku 4.5	Econômico	$1,00	$5,00
Anthropic	Claude Sonnet 4.6	Intermediário	$3,00	$15,00
Anthropic	Claude Opus 4.6	Premium	$5,00	$25,00
OpenAI	GPT-4o-mini	Econômico	$0,15	$0,60
OpenAI	GPT-4.1-mini	Econômico	$0,40	$1,60
OpenAI	GPT-4.1	Intermediário	$2,00	$8,00
OpenAI	GPT-5.4	Premium	$2,50	$15,00

Fonte: Documentação oficial dos provedores, verificada em abril de 2026.

2.4 Métricas de Avaliação

A comparação entre as abordagens foi feita com métricas clássicas de classificação (GOUTTE; GAUSSIER, 2005). Cada uma delas mede um aspecto diferente: Acurácia é a proporção de classificações corretas sobre o total; Precisão mede quantos dos alertas foram de fato ataques (verdadeiros positivos entre as predições positivas); Revocação ou Recall mede quantos ataques reais o sistema identificou corretamente (verdadeiros positivos entre os exemplos reais de uma classe); F1-Score é a média harmônica entre precisão e revocação, sendo útil quando se busca equilibrar ambos os tipos de erro; Macro-F1 calcula o F1 de cada classe separadamente e tira a média aritmética, tratando todas as classes com o mesmo peso, independentemente do tamanho; e a Matriz de Confusão cruza classificações reais e preditas, permitindo visualizar onde estão os erros. O Macro-F1 tem peso especial neste estudo porque as classes cmdi (89 registros, 0,3%) e path-traversal (290 registros, 0,9%) somam menos de

1,2% do dataset. Se olhássemos só para a acurácia, o desempenho inferior nessas classes minoritárias não seria evidenciado.

3. TRABALHOS RELACIONADOS

A detecção de ameaças em aplicações web é um tema bastante estudado, e a literatura pode ser organizada em três grandes linhas: métodos baseados em regras, aprendizado de máquina clássico e, mais recentemente, modelos de linguagem.

Na linha dos WAFs tradicionais, a referência principal é o OWASP Core Rule Set (CRS) para o ModSecurity, que mantém milhares de regras regex atualizadas por uma comunidade ativa (OWASP FOUNDATION, 2026; RISTIC, 2017). Apesar da ampla adoção, o CRS tem dois limites conhecidos: não cobre variantes de ataque que ainda não foram catalogadas e precisa de manutenção contínua para acompanhar novas técnicas de ofuscação. Esse ponto é discutido em detalhe por Fredj et al. (2021), em revisão abrangente sobre métodos de proteção de aplicações web.

Em paralelo, várias pesquisas aplicaram algoritmos de aprendizado de máquina à detecção de injeção SQL. Mehta et al. (2023) fizeram uma análise comparativa combinando métodos supervisionados e não supervisionados, enquanto Crespo-Martínez et al. (2023) propuseram detecção em fluxos de rede. Um trabalho particularmente relevante para esta pesquisa é o de Rashimo (2020), que aplicou redes neurais convolucionais sobre caracteres (ChCNN) exatamente no HttpParamsDataset empregado neste estudo. Os resultados evidenciaram a dificuldade em cenários desbalanceados: F1-Score próximo a 1,00 nas classes majoritárias, mas caindo para F1 ≈ 0,52 nas minoritárias.

Para XSS, a revisão sistemática de Kaur, Garg e Bathla (2023) sobre técnicas de aprendizado de máquina chega a uma conclusão semelhante: abordagens que usam features semânticas superam a correspondência por padrões puros, mas exigem elevada engenharia de atributos e têm dificuldade de generalizar para ataques inéditos.

O uso de LLMs em cibersegurança é um campo recente, que ganhou destaque na literatura nos últimos anos. Yao et al. (2024) publicaram um survey abrangente catalogando aplicações benéficas de LLMs nessa área, incluindo detecção de vulnerabilidades, análise de malware e triagem de incidentes. Zhou et al. (2025) fizeram uma revisão sistemática focada no uso de LLMs para detecção e reparo de vulnerabilidades, mostrando que modelos de

fronteira como GPT-4 e Claude têm desempenho superior às abordagens tradicionais em tarefas que exigem compreensão contextual. Xu et al. (2024) chegam a conclusões semelhantes em revisão que cobre múltiplos domínios da cibersegurança.

Diante do panorama apresentado, este estudo se posiciona como uma resposta à lacuna já apontada na Introdução e confirmada pela revisão aqui discutida: falta, na literatura, uma comparação sistemática entre o baseline regex e múltiplos LLMs, com diferentes modalidades de uso e rastreamento detalhado dos custos reais envolvidos. Os estudos revisados tendem a se concentrar em um único modelo ou a comparar LLMs entre si, sem confrontá-los com um baseline regex robusto nem explicitar o trade-off econômico entre as duas abordagens. É essa lacuna que este trabalho busca preencher, avaliando sete modelos de dois provedores, em duas modalidades, com validação estatística pareada e rastreamento detalhado de custos e tempo.

4. METODOLOGIA

4.1 Dataset

O dataset utilizado é o HttpParamsDataset (MORZEUX, 2016), de autoria do usuário Morzeux no GitHub, disponibilizado publicamente sob licença MIT. Foi criado para pesquisa em detecção de anomalias e contém valores que representam parâmetros de requisições HTTP. Na versão consolidada, são 31.067 registros distribuídos conforme a Tabela 2.

Tabela 2 — Distribuição das classes no dataset

Classe	Tipo	Quantidade	Percentual
norm	Tráfego normal	19.304	62,1%
sqli	Injeção de SQL	10.852	34,9%
xss	Cross-Site Scripting	532	1,7%
path-traversal	Travessia de diretório	290	0,9%
cmdi	Injeção de comandos	89	0,3%
Total	—	**31.067**	**100%**

Fonte: Elaboração própria com base no HttpParamsDataset (MORZEUX, 2016).

O desbalanceamento entre as classes é bastante marcado: a classe majoritária (norm) responde por 62,1% dos dados, enquanto cmdi representa apenas 0,3%. Esse perfil reflete o que se vê em ambientes reais de segurança, onde tráfego legítimo domina e certos tipos de ataque são raros.

4.2 Subamostragem e Reprodutibilidade

Submeter o dataset completo a cada chamada de API não é viável por dois motivos: primeiro, 31.067 registros excedem a janela de contexto prática de vários modelos; segundo, o custo de processar todo o conjunto múltiplas vezes seria proibitivo. Como alternativa, foram geradas 500 subamostras de 500 registros cada, com sementes reprodutíveis (seeds de 42 a 541). Dessas 500, foram selecionadas 30 para os experimentos — número suficiente para aplicar teste t pareado e calcular intervalos de confiança de 95% (MONTGOMERY, 2019).

Cada subamostra foi gerada em duas versões: uma rotulada, usada na avaliação e pelo motor de regras, e outra sem rótulo, enviada ao LLM, para evitar qualquer vazamento da resposta esperada.

4.3 Abordagens Avaliadas

Foram comparadas três abordagens de classificação, descritas nas subseções a seguir.

4.3.1 Motor de Regras Estático (Regex)

O motor baseado em regras é o baseline tradicional do estudo e simula um Web Application Firewall (WAF) simplificado. Foi implementado em Python, usando a biblioteca padrão `re`, com custo operacional zero por execução e latência sub-milissegundo por payload.

O conjunto é formado por 40 expressões regulares, distribuídas igualmente entre as quatro categorias de ataque (10 regras por categoria). As regras foram construídas a partir dos padrões sintáticos documentados pelo OWASP ModSecurity Core Rule Set (OWASP FOUNDATION, 2026) e pela literatura consolidada sobre detecção de ataques web (RISTIC, 2017; FREDJ et al., 2021). Não se trata de uma transcrição direta do CRS — que conta com milhares de regras mantidas pela comunidade —, mas de um conjunto reduzido que segue os mesmos princípios e cobre os padrões mais recorrentes para cada categoria, conforme detalhado nas subseções a seguir.

4.3.1.1 Regras para SQL Injection

As regras dessa categoria cobrem:

- Tautologias clássicas do tipo `OR 1=1` e variantes com operadores lógicos
- Comentários SQL (`--`, `#`, `/* */`)

- Palavras-chave de manipulação de dados (`UNION`, `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `DROP`)
- Acesso a metadados do banco (`information_schema`, `sysobjects`)
- Ataques time-based (`SLEEP`, `WAITFOR DELAY`, `BENCHMARK`)
- Funções de codificação muito usadas em ofuscação (`CHAR`, `ASCII`, `HEX`, `UNHEX`)

4.3.1.2 Regras para Cross-Site Scripting

As regras dessa categoria reconhecem:

- Tags `<script>` e suas variantes
- Handlers de eventos JavaScript (`onerror`, `onload`, `onclick`)
- Pseudo-protocolos (`javascript:`, `vbscript:`, `data:text/html`)
- Tags HTML frequentemente abusadas (`img`, `svg`, `iframe`, `body`, `object`)
- Chamadas a funções sensíveis (`alert()`, `eval()`, `document.cookie`)
- Payloads codificados em URL (`%3Cscript`)

4.3.1.3 Regras para Path Traversal

As regras dessa categoria cobrem:

- Sequências de travessia `../` e suas variantes codificadas (`%2e%2e%2f`, `%252e%252e`)
- Arquivos sensíveis em sistemas Unix (`/etc/passwd`, `/etc/shadow`, `/proc/self`) e Windows (`boot.ini`, `win.ini`, `web.config`)
- Esquemas de URI (`file://`)
- Caminhos absolutos (`C:\`, `/usr/`)

4.3.1.4 Regras para Command Injection

As regras dessa categoria identificam:

- Separadores de comando (`;`, `|`, `&&`, `||`)
- Binários tipicamente chamados em exploração (`ls`, `cat`, `whoami`, `wget`, `curl`, `nc`, `bash`)

- Substituição de comandos (`$(...)`, `` `...` ``)
- Redirecionamento de saída para diretórios sensíveis
- Utilitários de codificação (`base64 -d`, `xxd -r`)

4.3.1.5 Lógica de Classificação

A lógica de classificação é determinística e sequencial: para cada payload de entrada, o motor avalia as quatro categorias na ordem SQLi → XSS → Path Traversal → CMDi. Se alguma das 10 regras de uma categoria casa com o payload (via `re.search`), o motor retorna imediatamente o tipo de ataque correspondente, sem avaliar as categorias seguintes. Caso nenhuma das 40 regras seja ativada após percorrer todas as categorias, o payload é classificado como tráfego normal (`norm`).

A ordem de avaliação foi escolhida com base na prevalência decrescente das classes no dataset (Tabela 2): SQLi (34,9%) é verificada primeiro por ser, de longe, a categoria de ataque mais frequente, seguida por XSS (1,7%), Path Traversal (0,9%) e CMDi (0,3%). Essa estratégia de "primeiro casamento ganha" é a mesma adotada por WAFs comerciais quando configurados em modo de classificação única, e tem como vantagem o curto-circuito da avaliação: payloads claramente maliciosos são identificados sem necessidade de testar todas as 40 regras. Como contrapartida, em casos de conflito (um payload que poderia casar com regras de mais de uma categoria), o motor sempre prefere a categoria avaliada primeiro, o que pode introduzir vieses de classificação em payloads ambíguos.

4.3.1.6 Custos e Trade-offs da Implementação Manual

A construção do conjunto de regras envolveu duas etapas principais: (i) revisão da documentação do OWASP ModSecurity Core Rule Set e da literatura sobre cada categoria de ataque, para identificar os padrões sintáticos mais recorrentes; e (ii) tradução desses padrões em expressões regulares específicas, com testes iterativos contra exemplos do próprio dataset para ajustar a precisão e evitar falsos positivos óbvios. Mesmo para um conjunto reduzido como este, o esforço estimado foi de aproximadamente 12 a 15 horas de trabalho de uma pessoa com conhecimento prévio em segurança web e expressões regulares.

Em ambientes corporativos, esse custo escala de forma significativa. Conjuntos de regras de produção como o próprio CRS contam com milhares de regras mantidas por uma comunidade ativa de especialistas, com revisões e atualizações constantes para acompanhar a

evolução das técnicas de ataque. Mesmo equipes que apenas adotam o CRS sem escrevê-lo do zero precisam dedicar tempo recorrente a três atividades: (a) ajuste fino das regras para o contexto da aplicação protegida, evitando bloqueios indevidos de tráfego legítimo; (b) acompanhamento de novos vetores de ataque divulgados em fontes como CVE e bases de inteligência de ameaças; e (c) auditoria das regras existentes para retirada das que se tornaram obsoletas. Essas atividades exigem mão de obra especializada — analistas de segurança com experiência em desenvolvimento de regras —, profissionais cujo custo no mercado é elevado e cuja disponibilidade frequentemente representa um gargalo para a atualização tempestiva dos sistemas de defesa.

O código-fonte completo do motor, com as 40 regras documentadas e comentadas, está disponível no repositório do projeto (FONCECA, 2026), o que permite a reprodução integral dos experimentos e auditoria das expressões regulares utilizadas.

4.3.2 LLM com Análise Textual (llm_text)

Nesta modalidade, o modelo classifica os payloads usando apenas seu conhecimento sobre segurança, sem gerar nenhum código. O objetivo é avaliar a capacidade do LLM atuando como analista de segurança.

4.3.3 LLM com Geração de Regex (llm_regex)

Aqui, o modelo recebe a tarefa de gerar um script Python completo com regras regex a cada execução. O script é então executado localmente sobre os dados. Quando há falha de sintaxe ou erro de runtime, uma nova tentativa é feita com um prompt simplificado.

4.4 Pipeline de Avaliação

Para rodar todos os experimentos de forma consistente, foi desenvolvido um pipeline automatizado em Python, disponível publicamente (FONCECA, 2026). O pipeline implementa: (i) checkpoint por experimento individual, permitindo retomar a execução do ponto exato de interrupção — ao reexecutar o comando, os resultados já computados são detectados e pulados automaticamente; (ii) paralelização dos experimentos dentro de cada subamostra, via pool de threads com concorrência configurável, enquanto as subamostras são processadas sequencialmente; (iii) tratamento de erros de limite de taxa (HTTP 429) pelos mecanismos de retry fornecidos pelos SDKs oficiais dos provedores; (iv) mecanismo de retry com prompt simplificado nos casos em que o script regex gerado pelo LLM falha por erro de

sintaxe ou runtime; (v) rastreamento detalhado de tokens consumidos e custo financeiro por chamada; e (vi) temperatura fixada em 0 para todos os modelos, garantindo reprodutibilidade.

Os experimentos dos dois provedores foram executados em dois processos paralelos do pipeline, um para cada provedor: concorrência de 4 processos simultâneos para a Anthropic e 2 para a OpenAI, configurações ajustadas empiricamente de acordo com os limites de taxa observados em cada API. Entre os provedores não há dependência (APIs independentes), de modo que ambas as execuções rodam em paralelo entre si. Com essa estratégia, o tempo de parede efetivo da bateria completa de 450 pontos experimentais (30 subamostras × 15 configurações) foi de aproximadamente 3,5 horas, em contraste com as 9,8 horas de tempo computacional agregado reportadas na Seção 5.5.

4.5 Configuração dos Experimentos

No total, foram 15 experimentos: 1 baseline regex e 14 configurações baseadas em LLM (7 modelos × 2 abordagens). Cada um deles foi executado sobre as mesmas 30 sub amostragens, o que resulta em 450 avaliações individuais.

4.6 Execução e Desafios Operacionais

A execução completa da bateria experimental foi organizada em três etapas, conforme documentado no repositório do projeto (FONCECA, 2026). A primeira consiste em executar os experimentos da Anthropic e da OpenAI em dois processos paralelos independentes, um por provedor, cada qual com sua própria configuração de concorrência (4 processos simultâneos para a Anthropic e 2 para a OpenAI). Esses valores foram ajustados empiricamente após observação dos limites de taxa praticados por cada provedor no tier de API utilizado: a OpenAI passou a retornar erros HTTP 429 (too many requests) com frequência elevada quando a concorrência ultrapassava 2 processos, enquanto a Anthropic suportou estavelmente até 4 processos simultâneos.

A segunda etapa, automática, consiste no salvamento incremental de cada resultado em arquivos de checkpoint individuais por experimento e subamostra. Esse mecanismo se mostrou crítico durante a execução: ocorreram interrupções pontuais por timeouts de rede, esgotamento momentâneo de cota da API e instabilidades transitórias de conectividade. Sem o checkpoint, qualquer interrupção exigiria reinício completo da bateria, com desperdício significativo de tempo e custo financeiro — visto que tokens já consumidos não são

reembolsáveis. Com a estratégia adotada, a retomada ocorre exatamente no ponto de parada, e os experimentos já concluídos são detectados e ignorados automaticamente em uma nova execução do mesmo comando.

A terceira etapa consiste na mesclagem dos checkpoints dos dois provedores em um conjunto unificado, a partir do qual são geradas as métricas consolidadas, as figuras e as tabelas comparativas apresentadas na Seção 5. Toda essa orquestração — incluindo os comandos completos, os parâmetros de concorrência e o tratamento de retentativas — está documentada no repositório público (FONCECA, 2026), o que permite a reprodução integral dos experimentos por terceiros.

5. RESULTADOS E DISCUSSÃO

5.1 Desempenho Geral

A Tabela 3 apresenta os resultados consolidados. As células verdes indicam os melhores resultados e as vermelhas, os piores. Os valores são médias das 30 sub amostragens.

Tabela 3 — Resultados consolidados (média de 30 subamostragens). ★ = melhor resultado, † = pior resultado

Experimento	Acc	Macro-Prec	Macro-Rec	Macro-F1	Custo (USD)
Regex (estático)	0,976	0,940	0,828	0,867	$0,00
Haiku 4.5 — texto	0,999	0,969	0,967	**★ 0,967**	$1,96
Haiku 4.5 — regex	0,916	0,485	0,532	0,425	$0,21
Sonnet 4.6 — texto	0,999	0,965	0,962	0,963	$5,88
Sonnet 4.6 — regex	0,970	0,813	0,690	0,724	$2,08
Opus 4.6 — texto	**★ 0,999**	**★ 0,971**	**★ 0,966**	**★ 0,968**	$9,80
Opus 4.6 — regex	0,971	0,854	0,755	0,783	$5,47
GPT-4o-mini — texto	0,939	0,761	0,815	0,771	$0,18
GPT-4o-mini — regex	† 0,626	† 0,125	† 0,200	† 0,154	$0,19
GPT-4.1-mini — texto	0,995	0,911	0,906	0,904	$0,49
GPT-4.1-mini — regex	0,928	0,779	0,583	0,620	$0,21
GPT-4.1 — texto	0,999	0,966	0,965	0,965	$2,53
GPT-4.1 — regex	0,961	0,786	0,676	0,644	$1,85
GPT-5.4 — texto	0,998	0,958	0,938	0,946	$4,20
GPT-5.4 — regex	0,928	0,829	0,788	0,764	$0,77

Fonte: Elaboração própria.

Três pontos se destacam nos resultados. O primeiro é que praticamente todos os modelos LLM na modalidade textual superam o motor de regras em Macro-F1 — a única exceção é o GPT-4o-mini. O segundo é que o melhor resultado absoluto (Opus 4.6 texto, Macro-F1 = 0,968) fica quase empatado com o modelo mais econômico da mesma família (Haiku 4.5 texto, 0,967), o que sugere que, para esta tarefa, o uso do modelo premium pode não se justificar. O terceiro, e mais contra intuitivo, é que nenhum LLM conseguiu superar as regras estáticas na modalidade de geração de regex.

A Figura 1 ilustra a comparação do Macro-F1 entre todos os experimentos.

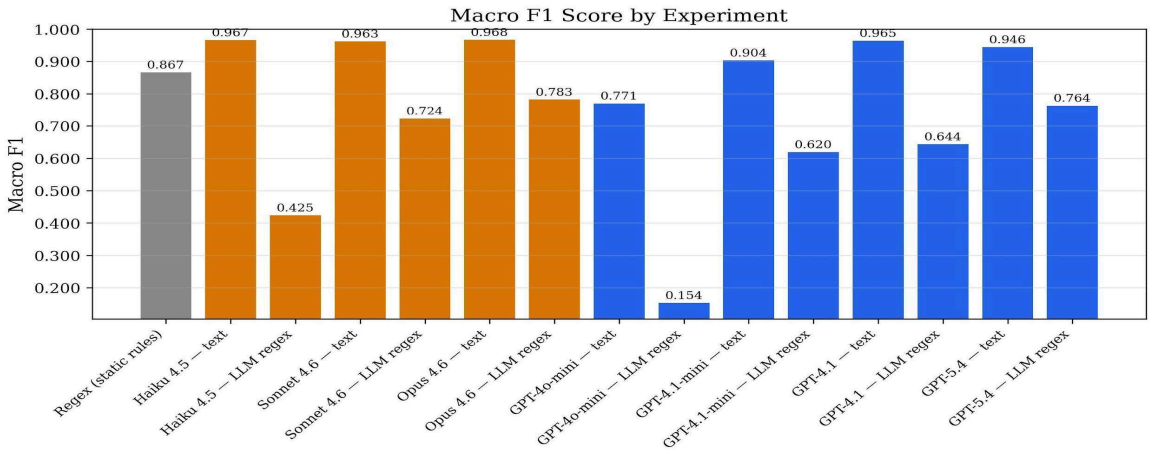

Figura 1 — Comparação do Macro-F1. Barras cinzas = regex, laranjas = Anthropic, azuis = OpenAI.

Fonte: Elaboração própria.

5.2 Análise por Classe de Ataque

A Tabela 4 detalha o F1-Score por classe, evidenciando onde cada abordagem falha.

Tabela 4 — F1-Score por classe de ataque (★ = melhor, † = pior)

Experimento	F1 norm	F1 sqli	F1 cmdi	F1 xss	F1 path-trav.
Regex (estático)	0,983	0,969	0,611	0,933	0,840
Haiku 4.5 — texto	0,999	0,999	**★ 0,855**	**0,988**	**★ 0,993**
Haiku 4.5 — regex	0,993	0,909	0,076	† 0,000	0,146
Sonnet 4.6 — texto	**★ 1,000**	**★ 1,000**	0,827	**★ 0,997**	**★ 0,993**
Opus 4.6 — texto	**★ 1,000**	**★ 1,000**	**0,853**	**★ 0,997**	0,990
GPT-4o-mini — texto	0,955	0,926	0,597	0,591	0,785
GPT-4o-mini — regex	† 0,769	† 0,000	† 0,000	† 0,000	† 0,000
GPT-4.1 — texto	0,999	0,999	0,844	0,988	0,993
GPT-5.4 — texto	0,999	0,999	0,827	0,989	0,916

Fonte: Elaboração própria. Exibidos os experimentos mais relevantes.

O maior diferencial está na classe cmdi: enquanto o motor de regras atinge F1 de 0,611, os melhores LLMs chegam a 0,855 — um ganho de cerca de 40%. A classe xss tem o exemplo mais extremo: o Haiku regex tem F1 de 0,000 (falha total), mas o mesmo modelo na modalidade textual alcança 0,988, com custo similar. Isso mostra que a principal capacidade do LLM nessa tarefa não está na produção de padrões, e sim na compreensão semântica já incorporada em seu treinamento.

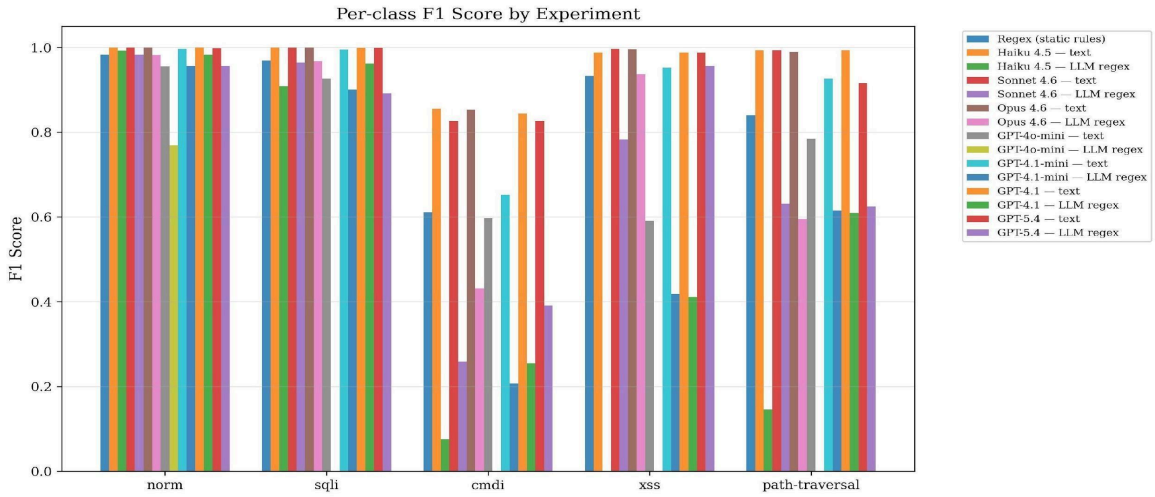

Figura 2 — F1-Score por classe de ataque e experimento.

Fonte: Elaboração própria.

5.3 Matrizes de Confusão

As matrizes de confusão agregadas ao longo das 30 sub amostragens (15.000 predições em cada) deixam claros os padrões de erro de cada abordagem.

A Figura 3 mostra que o motor de regras deixa de identificar 322 casos maliciosos: 275 payloads SQLi (5,3%) e 20 cmdi (34,5%) que foram classificados como tráfego normal — ou seja, ataques que um WAF baseado nessas regras não sinalizaria.

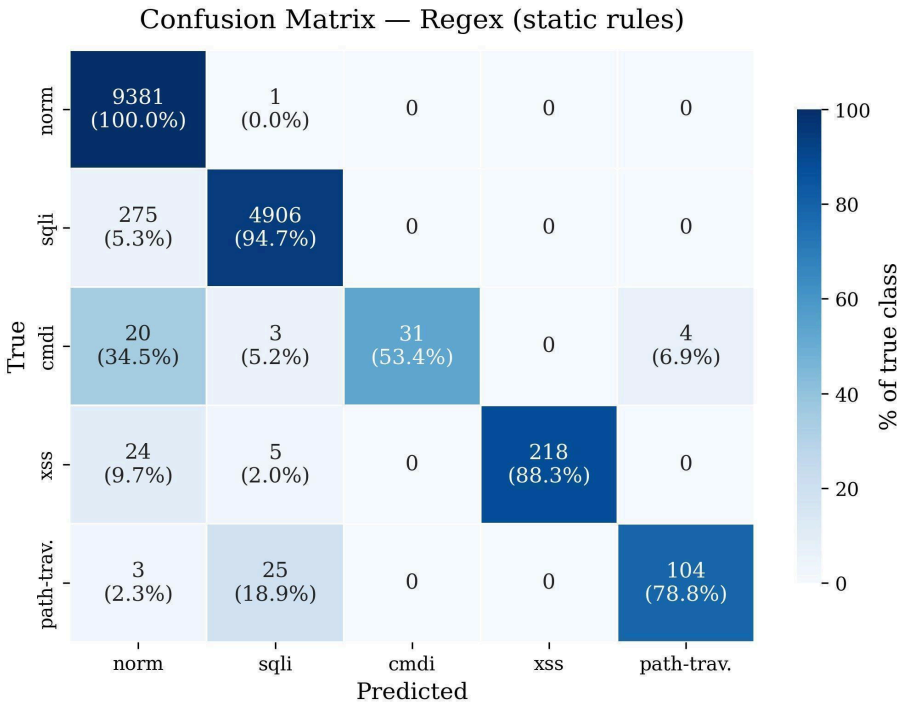

Figura 3 — Matriz de confusão: motor de regras. Destaque: 275 SQLi e 20 cmdi classificado como normal.

Fonte: Elaboração própria.

A Figura 4 mostra o cenário oposto: o Claude Haiku 4.5 texto comete apenas 11 erros em 15.000 predições. A classe cmdi, detectada em 53,4% pelo regex, sobe para 98,3% com o Haiku — uma redução de 96,6% nos falsos negativos.

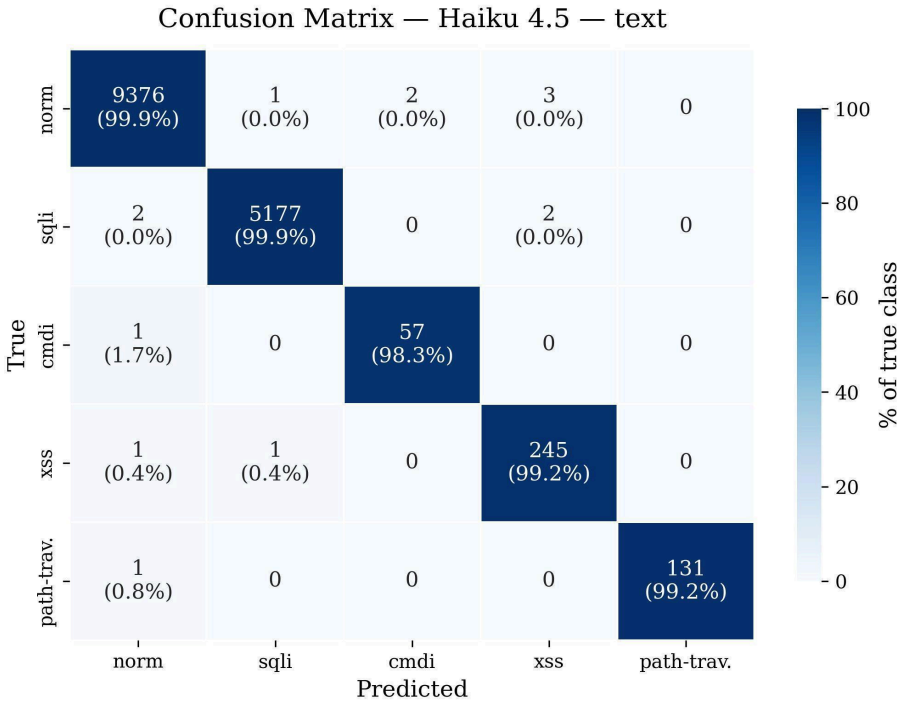

Figura 4 — Matriz de confusão: Claude Haiku 4.5 texto. Apenas 11 erros em 15.000 predições.

Fonte: Elaboração própria.

A Figura 5 documenta o pior caso do estudo: o GPT-4o-mini em modo regex falhou totalmente, classificando todos os 5.618 payloads maliciosos como normais. O script gerado pelo modelo não trata o cabeçalho do CSV, resultando na classificação de todos os registros como tráfego legítimo.

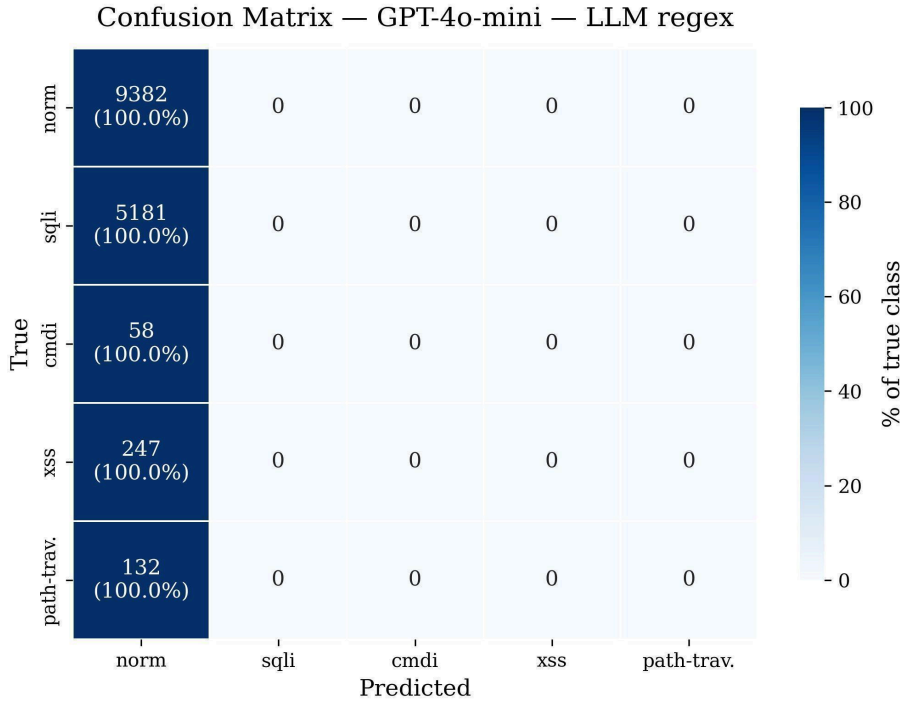

Figura 5 — Matriz de confusão: GPT-4o-mini regex — falha total (100% dos ataques não detectados).

Fonte: Elaboração própria.

5.4 Comparação Estatística

Os testes t pareados confirmam que a superioridade da análise textual não é resultado de variabilidade das sub amostragens: em 5 dos 7 modelos, a diferença é estatisticamente significativa com $p < 0{,}0001$. A Tabela 5 resume os resultados.

Tabela 5 — Testes t pareados vs. motor de regras (Macro-F1). Diferença positiva = LLM superior.

Experimento	Δ Macro-F1	Estatística t	Valor p	Sig.
Haiku 4.5 — texto	**+0,100**	**7,63**	**< 0,0001**	*
Opus 4.6 — texto	**+0,101**	**7,49**	**< 0,0001**	*
GPT-4.1 — texto	**+0,098**	**7,50**	**< 0,0001**	*
GPT-5.4 — texto	**+0,078**	**4,10**	**< 0,0001**	*
GPT-4.1-mini — texto	+0,037	1,67	0,0940	n.s.
GPT-4o-mini — texto	−0,096	−3,43	0,0006	*†

Haiku 4.5 — regex	−0,443	−28,75	< 0,0001	*†
Opus 4.6 — regex	−0,084	−5,38	< 0,0001	*†
GPT-4o-mini — regex	−0,713	−48,35	< 0,0001	*†

*Fonte: Elaboração própria. * = p < 0,05. † = significativamente pior que o baseline. n.s. = não significativo.*

Destaca-se que o GPT-4.1-mini texto, embora apresente Macro-F1 superior ao regex (0,904 vs 0,867), não alcança significância estatística (p = 0,094). Já o GPT-4o-mini texto é significativamente pior que o baseline, sendo o único modelo a regredir na modalidade textual.

5.5 Análise de Custos e Tempo

A Tabela 6 detalha custos e tempo com totais por provedor — informação essencial para decisões de implantação em produção.

Tabela 6 — Custos e tempo por experimento e por provedor (30 sub amostragens)

Experimento	Tok. entrada	Tok. saída	Custo (USD)	Tempo
Regex (estático)	—	—	**$0,00**	**0,1s**
Haiku 4.5 — texto	329.478	326.171	$1,96	20,9 min
Haiku 4.5 — regex	9.030	40.702	$0,21	5,1 min
Sonnet 4.6 — texto	329.508	326.091	$5,88	41,7 min
Sonnet 4.6 — regex	19.134	134.805	$2,08	34,7 min
Opus 4.6 — texto	329.508	326.116	$9,80	56,2 min
Opus 4.6 — regex	9.060	216.916	$5,47	44,3 min
Subtotal Anthropic	—	—	**$25,40**	**3,4 h**
GPT-4o-mini — texto	296.224	232.796	$0,18	80,7 min
GPT-4o-mini — regex	20.100	304.341	$0,19	88,7 min
GPT-4.1-mini — texto	296.224	230.152	$0,49	57,2 min
GPT-4.1-mini — regex	18.516	127.703	$0,21	28,1 min
GPT-4.1 — texto	296.224	242.711	$2,53	32,6 min
GPT-4.1 — regex	12.972	227.638	$1,85	36,5 min
GPT-5.4 — texto	296.194	230.474	$4,20	45,4 min
GPT-5.4 — regex	8.190	49.678	$0,77	13,5 min
Subtotal OpenAI	—	—	**$10,42**	**6,4 h**
TOTAL GERAL	—	—	**$35,81**	**9,8 h**

Fonte: Elaboração própria.

Observa-se que a Anthropic responde por 70,9% do custo total (US$ 25,40 de US$ 35,81), mas o tempo agregado de seus experimentos é de apenas 3,4 horas contra 6,4 horas da

OpenAI. Isso se explica por duas razões: os modelos da Anthropic cobram mais por token, mas respondem com mais rapidez e enfrentam menos throttling (limites de taxa). Vale notar que os tempos reportados na Tabela 6 correspondem à soma linear dos tempos individuais de cada experimento (tempo computacional agregado); graças à paralelização descrita na Seção 4.4 — 4 processos simultâneos para Anthropic, 2 para OpenAI e ambos os provedores rodando em paralelo entre si —, o tempo de parede efetivo da bateria completa foi de aproximadamente 3,5 horas.

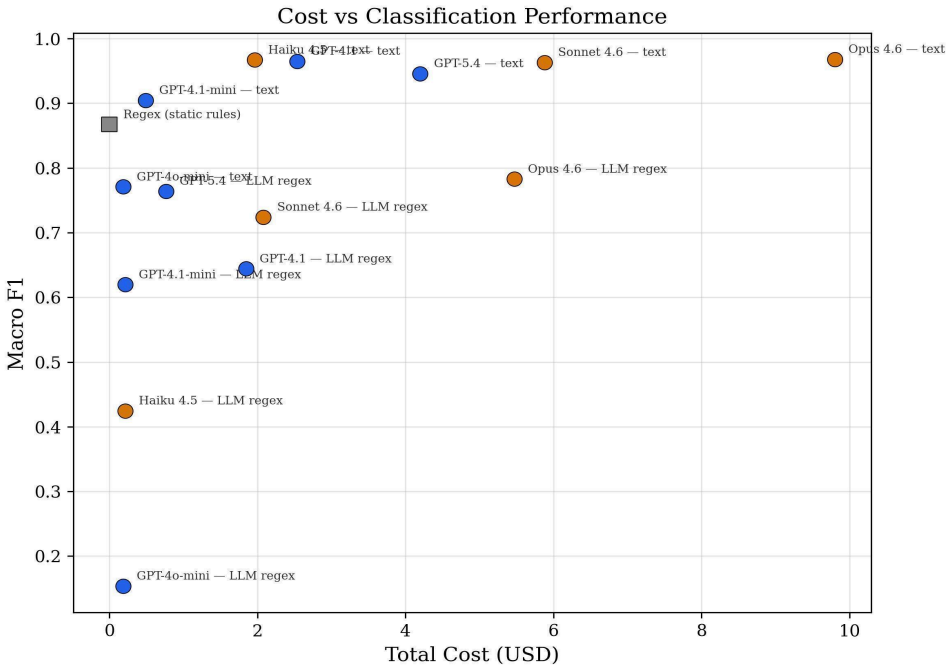

Figura 6 — Relação custo-benefício: custo total vs. Macro-F1. O quadrante superior esquerdo (alto F1, baixo custo) é ideal.

Fonte: Elaboração própria.

A Figura 6 evidencia o Claude Haiku 4.5 texto como ponto ótimo de custo-benefício: Macro-F1 de 0,967 por US$ 1,96. O GPT-4.1 texto (0,965 por US$ 2,53) é a alternativa mais próxima. Se o objetivo for custo mínimo, o motor de regras continua sendo uma opção viável (0,867 por US$ 0,00), mas com a limitação de deixar 322 ataques em 15.000 sem detecção.

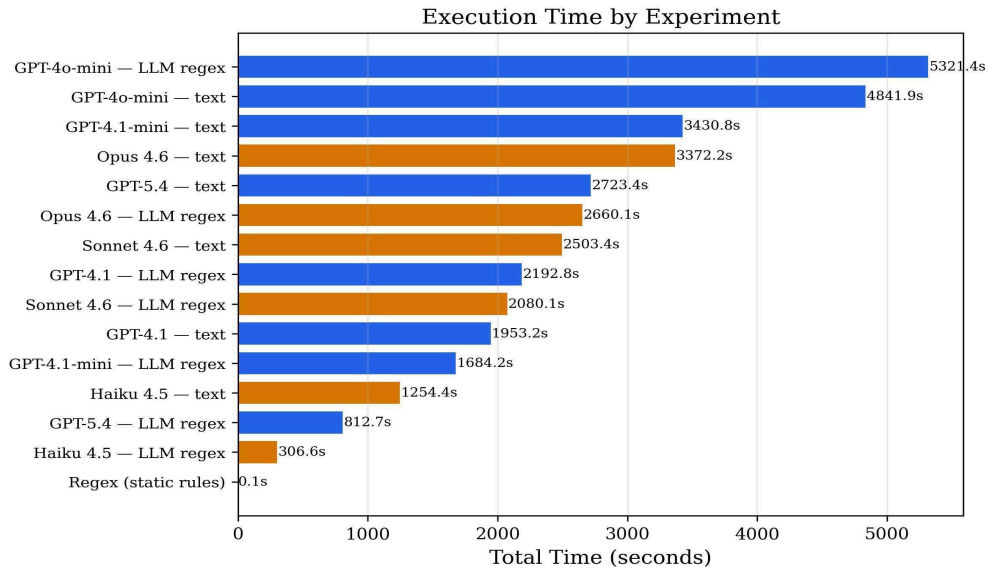

Figura 7 — Tempo de execução total por experimento (30 sub amostragens).

Fonte: Elaboração própria.

5.6 Análise Textual vs. Geração de Regex

O achado mais surpreendente deste estudo é que a geração de scripts regex pelos LLMs é inferior à análise textual direta e, na maioria dos casos, inferior até ao motor de regras escrito manualmente.

Tabela 7 — Comparação entre modalidades por modelo: texto vs. regex gerado

Modelo	F1 texto	F1 regex	Perda	Custo texto	Custo regex
Claude Haiku 4.5	**0,967**	0,425	−0,542	$1,96	$0,21
Claude Sonnet 4.6	0,963	0,724	−0,239	$5,88	$2,08
Claude Opus 4.6	0,968	0,783	−0,185	$9,80	$5,47
GPT-4o-mini	0,771	0,154	−0,617	$0,18	$0,19
GPT-4.1-mini	0,904	0,620	−0,284	$0,49	$0,21
GPT-4.1	0,965	0,644	−0,321	$2,53	$1,85
GPT-5.4	0,946	0,764	−0,182	$4,20	$0,77

Fonte: Elaboração própria.

O padrão de tokens ajuda a entender o que está acontecendo. Na modalidade textual, entrada e saída ficam equilibradas (cerca de 330 mil tokens cada, no caso da Anthropic). Já na geração de regex, o Opus produz 24 vezes mais tokens de saída do que de entrada (216 mil contra 9 mil), escrevendo scripts extensos. Entretanto, esses scripts sofisticados falham com frequência: o Sonnet 4.6 usou de forma sistemática o flag re.VERBOSE com padrões multilinha e gerou erros de sintaxe em 100% das tentativas iniciais (corrigidos pelo

mecanismo de retry); o GPT-4o-mini produziu scripts que não conseguiam tratar o cabeçalho do CSV em todas as 30 sub amostragens; e o GPT-4.1 chegou a gerar scripts com mais de 800 linhas, cujos próprios padrões regex ultrapassaram a capacidade do modelo de manter consistência ao longo do script.

Desse resultado deriva uma implicação prática relevante: para classificação de segurança, é mais eficaz utilizar o LLM diretamente como classificador do que pedir que ele produza código para a tarefa. Esse padrão é coerente com estudos que mostram que LLMs frequentemente geram código com falhas funcionais (LIU et al., 2023) ou inseguro (PEARCE et al., 2022).

Vale destacar que essa degradação não é uniforme entre modelos: os econômicos apresentam quedas substancialmente maiores que os premium. Enquanto o Opus 4.6 e o GPT-5.4 mantêm Macro-F1 acima de 0,76 na geração de regex, o GPT-4o-mini cai para F1 = 0,154 e o Haiku 4.5 para F1 = 0,425, esse último com falha total na classe XSS. As matrizes de confusão revelam que essas falhas decorrem não apenas de regras imprecisas, mas de defeitos estruturais no código gerado — como tratamento incorreto do cabeçalho do CSV no caso do GPT-4o-mini. Esse padrão sugere que a tarefa exige duas capacidades simultâneas (conhecimento semântico de ataques e competência em programação), e que modelos econômicos atendem melhor à primeira do que à segunda. A implicação prática é que, ao optar por modelos econômicos para reduzir custos, a modalidade textual direta é uma escolha mais segura do que a delegação da geração de regras.

5.7 Comparação entre Provedores: Anthropic vs. OpenAI

A Tabela 8 sintetiza a comparação direta entre os provedores.

Tabela 8 — Comparação direta Anthropic vs. OpenAI

Aspecto	Anthropic	OpenAI
Melhor modelo (texto)	**Opus 4.6: F1=0,968**	GPT-4.1: F1=0,965
Melhor custo-benefício	**Haiku: F1=0,967 / $1,96**	GPT-4.1-mini: F1=0,904 / $0,49
Pior resultado regex	Haiku: F1=0,425	GPT-4o-mini: F1=0,154
Custo total (30 subs)	$25,40	**$10,42**
Tempo total	3,4 horas	6,4 horas
Erros totais (agregado)	8 em 15.000	15.832 em 15.000*

*Fonte: Elaboração própria. * Soma de erros de todos os experimentos do provedor. Inclui as 15.000 falhas do GPT-4o-mini regex (modelo classificou 100% dos payloads como tráfego normal).*

A Anthropic se destaca em qualidade: seus três modelos de texto ocupam os três primeiros lugares em Macro-F1. A OpenAI é mais econômica em custo absoluto (US$ 10,42 vs US$ 25,40), mas a combinação de custo e desempenho favorece o Haiku, que por US$ 1,96 entrega resultado comparável ao Opus de US$ 9,80 e superior ao GPT-5.4 de US$ 4,20.

Em tempo de resposta, a Anthropic é significativamente mais rápida (3,4h vs 6,4h para as mesmas 30 sub amostragens), o que se explica por menor latência média por requisição e por limites de taxa mais permissivos no tier utilizado, suportando concorrência de 4 processos simultâneos sem rejeições frequentes. A OpenAI apresentou o maior tempo de processamento individual no GPT-4o-mini (80,7 min para texto e 88,7 min para regex), provavelmente devido aos limites de taxa mais restritivos no tier de API utilizado, que exigiram concorrência reduzida a 2 processos simultâneos.

6. CONCLUSÃO

Este estudo apresentou uma comparação sistemática entre abordagens baseadas em regras e modelos de linguagem de grande escala para a classificação de payloads HTTP maliciosos. Ao todo, foram 15 configurações experimentais, cobrindo 7 modelos de 2 provedores, executadas sobre 30 sub amostragens com validação estatística pareada. O custo total ficou em US$ 35,81 (US$ 25,40 na Anthropic e US$ 10,42 na OpenAI), com 9,8 horas de tempo computacional agregado, reduzidas a aproximadamente 3,5 horas de tempo real graças à paralelização adotada.

Os achados principais podem ser resumidos em quatro pontos:

(1) A análise textual via LLM supera o motor de regras estáticas de forma significativa, ganhando até 10 pontos percentuais em Macro-F1 ($p < 0,0001$). Esse ganho aparece de forma mais concentrada nas classes minoritárias — cmdi (+40%) e path-traversal (+18%) —, que são justamente as mais prejudicadas pelas regras estáticas.

(2) Há uma inversão contraintuitiva quando se pede ao LLM para gerar regex: todos os modelos produzem scripts piores que o motor escrito à mão. O LLM apresenta melhor desempenho quando aplicado diretamente ao seu conhecimento, em vez de tentar codificá-lo em regras.

(3) Na família da Anthropic, o modelo mais econômico (Haiku 4.5, US$ 1,96) praticamente empata com o premium (Opus 4.6, US$ 9,80) nesta tarefa, o que sugere que modelos maiores não se justificam para classificação de payloads.

(4) A Anthropic se destaca em qualidade; a OpenAI, em custo por token. A escolha entre uma e outra depende da ponderação que cada equipe atribui em seu contexto.

Do ponto de vista prático, os resultados sugerem que sistemas híbridos — nos quais regras regex atuam como primeira linha de filtragem de baixo custo e LLMs são invocados para casos ambíguos ou suspeitos — podem oferecer o melhor equilíbrio entre desempenho, custo operacional e tempo de resposta em cenários de produção.

Como limitações, o estudo se baseia em um único dataset (HttpParamsDataset) e reflete os modelos e a precificação das APIs no momento da coleta (abril de 2026); novos modelos lançados posteriormente podem alterar o cenário observado. Como trabalhos futuros, são promissoras: a avaliação em datasets complementares com ataques mais recentes; a implementação e validação empírica de um sistema híbrido regex + LLM; e a investigação de fine-tuning específico em modelos econômicos para mitigar suas limitações na geração de regex.

O código-fonte completo está disponível publicamente (FONCECA, 2026) em: *https://github.com/flafonceca/llm-vs-regex-threat-detection.*

REFERÊNCIAS

ANTHROPIC. **Claude API documentation and pricing**. San Francisco: Anthropic, 2026. Disponível em: https://docs.anthropic.com. Acesso em: 07 abr. 2026.

APPELT, D.; NGUYEN, C. D.; PANICHELLA, A.; BRIAND, L. C. A machine-learning-driven evolutionary approach for testing web application firewalls. IEEE Transactions on Reliability, v. 67, n. 3, p. 733-757, 2018. DOI: 10.1109/TR.2018.2805763.

CRESPO-MARTÍNEZ, I. S. et al. SQL injection attack detection in network flow data. **Computers & Security**, v. 127, p. 103093, 2023. DOI: 10.1016/j.cose.2023.103093.

FONCECA, F. T. llm-vs-regex-threat-detection: pipeline de avaliação comparativa para classificação de payloads HTTP. [*S. l.*]: GitHub, 2026. Disponível em: https://github.com/flafonceca/llm-vs-regex-threat-detection. Acesso em: 17 abr. 2026.

FREDJ, O. B. et al. An OWASP top ten driven survey on web application protection methods. In: *Risks and Security of Internet and Systems — CRiSIS 2020*. Lecture Notes in Computer Science. Cham: Springer, 2021. v. 12528, p. 235-252.

GOUTTE, C.; GAUSSIER, E. A probabilistic interpretation of precision, recall and F-score, with implication for evaluation. In: *Advances in Information Retrieval — ECIR 2005*. Berlin: Springer, 2005. p. 345-359.

HALFOND, W. G. J.; VIEGAS, J.; ORSO, A. A classification of SQL-injection attacks and countermeasures. In: Proceedings of the International Symposium on Secure Software Engineering. Washington, D.C., USA: [s. n.], 2006.

HYDARA, I.; SULTAN, A. B. M.; ZULZALIL, H.; ADMODISASTRO, N. Current state of research on cross-site scripting (XSS): a systematic literature review. Information and Software Technology, v. 58, p. 170-186, 2015. DOI: 10.1016/j.infsof.2014.07.010.

KAUR, J.; GARG, U.; BATHLA, G. Detection of cross-site scripting (XSS) attacks using machine learning techniques: a review. **Artificial Intelligence Review**, v. 56, p. 12725-12769, 2023. DOI: 10.1007/s10462-023-10433-3.

LIU, J.; XIA, C. S.; WANG, Y.; ZHANG, L. Is your code generated by ChatGPT really correct? Rigorous evaluation of large language models for code generation. In: Advances in Neural Information Processing Systems (NeurIPS), v. 36, 2023.

MEHTA, D. et al. SQLIML: a comprehensive analysis for SQL injection detection using multiple supervised and unsupervised learning schemes. **SN Computer Science**, v. 4, n. 3, p. 281, 2023. DOI: 10.1007/s42979-022-01626-8.

MITRE CORPORATION. Common Weakness Enumeration (CWE). Bedford: The MITRE Corporation, 2024. Disponível em: https://cwe.mitre.org/. Acesso em: 22 abr. 2026.

MONTGOMERY, D. C. **Design and analysis of experiments**. 10. ed. Hoboken: Wiley, 2019.

MORZEUX. HttpParamsDataset: dataset contains several benign and attacks samples which can be used as values in HTTP protocol. [*S. l.*]: GitHub, 16 mar. 2016. Licença MIT. Disponível em: https://github.com/Morzeux/HttpParamsDataset. Acesso em: 17 abr. 2026.

OPENAI. **OpenAI API documentation and pricing**. San Francisco: OpenAI, 2026. Disponível em: https://platform.openai.com/docs. Acesso em: 07 abr. 2026.

OWASP FOUNDATION. A01:2021 – Broken Access Control. *OWASP Top 10:2021*. Disponível em: https://owasp.org/Top10/2021/A01_2021-Broken_Access_Control/. Acesso em: 15 abr. 2026.

OWASP FOUNDATION. A03:2021 – Injection. *OWASP Top 10:2021*. Disponível em: https://owasp.org/Top10/2021/A03_2021-Injection/. Acesso em: 15 abr. 2026.

OWASP FOUNDATION. OWASP CRS — Core Rule Set. Versão 4.25.0 LTS. [*S. l.*]: OWASP, 2026. Disponível em: https://coreruleset.org/. Acesso em: 17 abr. 2026.

PEARCE, H.; AHMAD, B.; TAN, B.; DOLAN-GAVITT, B.; KARRI, R. Asleep at the keyboard? Assessing the security of GitHub Copilot's code contributions. In: Proceedings of the 43rd IEEE Symposium on Security and Privacy (SP). San Francisco: IEEE, 2022. p. 754-768. DOI: 10.1109/SP46214.2022.9833571.

RASHIMO. ChCNN: a convolutional neural network approach to classify web requests. [S. l.]: GitHub, 2020. Disponível em: https://github.com/rashimo/ChCNN. Acesso em: 07 abr. 2026.

RISTIC, I. **ModSecurity handbook**: the complete guide to the popular open source web application firewall. 2. ed. London: Feisty Duck, 2017.

VASWANI, A. et al. Attention is all you need. In: *Advances in Neural Information Processing Systems (NeurIPS)*, v. 30, p. 5998-6008, 2017.

XU, H. et al. Large language models for cyber security: a systematic literature review. *arXiv preprint*, arXiv:2405.04760, 2024.

YAO, Y. et al. A survey on large language model (LLM) security and privacy: the good, the bad, and the ugly. **High-Confidence Computing**, v. 4, n. 2, p. 100211, 2024. DOI: 10.1016/j.hcc.2024.100211.

ZHOU, X. et al. Large language model for vulnerability detection and repair: literature review and the road ahead. **ACM Transactions on Software Engineering and Methodology**, v. 34, n. 5, 2025. DOI: 10.1145/3708522.