UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

DANIELA ARALDI

ANÁLISE DE REGRAS *SIGMA* GERADAS POR AGENTE AUTÔNOMO COM RAG

PATO BRANCO

2026

DANIELA ARALDI

ANÁLISE DE REGRAS *SIGMA* GERADAS POR AGENTE AUTÔNOMO COM RAG

Analyzing Sigma Rules Generated by a RAG-based Autonomous Agent

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia de Computação do Curso de Bacharelado em Engenharia de Computação da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Me. Adriano Serckumecka

Coorientador: Prof. Dr. Jefferson Tales Oliva

PATO BRANCO

2026

DANIELA ARALDI

ANÁLISE DE REGRAS *SIGMA* GERADAS POR AGENTE AUTÔNOMO COM RAG

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia de Computação do Curso de Bacharelado em Engenharia de Computação da Universidade Tecnológica Federal do Paraná.

Data de aprovação: 26/junho/2026

Adriano Serckumecka
Prof. Me.
Universidade Tecnológica Federal do Paraná

Jefferson Tales Oliva
Prof. Dr.
Universidade Tecnológica Federal do Paraná

Luis Cassiano Goularte Rista
Prof. Dr.
Universidade Tecnológica Federal do Paraná

Luiz Fernando Puttow Southier
Prof. Dr.
Universidade Tecnológica Federal do Paraná

PATO BRANCO
2026

Para meu pai (*em memória*).

AGRADECIMENTOS

Agradeço a todas as pessoas que conheci graças à graduação e cujos atravessamentos foram positivos, mas principalmente às amizades que permaneceram.

Agradeço aos professores que me inspiraram.

Agradeço aos professores que me orientaram, por se porem disponíveis a me orientar, e a todos os professores que foram compreensivos às sensibilidades humanas que um aluno carrega.

Agradeço aos meus pais, por me proporcionarem o privilégio de frequentar a universidade e sempre me apoiarem.

Agradeço às minhas amizades, por tornarem a vida mais que suportável.

Agradeço ao meu companheiro por acreditar em mim e me incentivar a finalizar o curso (me mandando ir pra terapia).

E agradeço substancialmente a mim mesma, por não ter desistido.

RESUMO

A crescente sofisticação e o volume das ameaças cibernéticas pressionam as equipes de segurança a reduzir o intervalo entre a divulgação de uma vulnerabilidade e a disponibilização da detecção correspondente. As regras Sigma constituem um padrão aberto para descrever lógicas de detecção em sistemas de gerenciamento de informações e eventos de segurança (SIEMs), mas sua escrita manual é lenta e propensa a lacunas diante de ameaças emergentes. Este trabalho desenvolve e avalia um agente autônomo para a geração automatizada de regras Sigma a partir de fontes diversificadas de inteligência de ameaças, com o propósito de apoiar o analista de segurança, a quem cabe a validação final. O agente é implementado como uma máquina de estados orientada a grafo que integra geração aumentada por recuperação (RAG) sobre um repositório de regras de referência, geração por um modelo de linguagem de grande escala (LLM) e validação iterativa da saída. Sua eficácia é avaliada por comparação com regras originais escritas por especialistas e por abordagens intermediárias. Os resultados sustentam a viabilidade do agente exclusivamente como ferramenta de apoio ao especialista. O desempenho observado, no entanto, não supera o de LLMs comerciais empregadas de forma direta, evidenciando que a arquitetura proposta, isoladamente, ainda não é competitiva para uso autônomo.

Palavras-chave: segurança da informação; detecção; agente; automação.

ABSTRACT

The increasing sophistication and volume of cyber threats pressure security teams to reduce the interval between the disclosure of a vulnerability and the availability of the corresponding detection. Sigma rules constitute an open standard for describing detection logic in Security Information and Event Management (SIEM) systems, but their manual authoring is slow and prone to gaps in the face of emerging threats. This work develops and evaluates an autonomous agent for the automated generation of Sigma rules from diverse threat intelligence sources, with the purpose of supporting the security analyst, who remains responsible for the final validation. The agent is implemented as a graph-oriented state machine that integrates retrieval-augmented generation (RAG) over a reference rule repository, rule generation by a large language model (LLM), and iterative validation of the output. Its effectiveness is assessed by comparison with expert-written reference rules and with intermediate approaches. The results support the feasibility of the agent exclusively as a supporting tool for the specialist. However, the observed performance does not surpass that of commercial LLMs used directly, evidencing that the proposed architecture, in isolation, is not yet competitive for autonomous use.

Keywords: information security; detection; agent; automation.

LIST OF ALGORITHMS

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

Siglas

API	*Application Programming Interface*, em português Interface de Programação de Aplicações
BGL	*BlueGene/L*
CAPEC	Common Attack Pattern Enumerations and Classifications
CISA	Agência de Segurança de Infraestrutura e Cibernética, do inglês *Cybersecurity and Infrastructure Security Agency*
CNN	Rede Neural Convolucional, do inglês *Convolutional Neural Network*
CoT	*Chain-of-Thought*
CVE	Vulnerabilidades e Exposições Comuns, do inglês *Common Vulnerabilities and Exposures*
CVSS	Sistema de Pontuação de Vulnerabilidades Comuns, do inglês *Common Vulnerability Scoring System*
CWE	Enumeração de Vulnerabilidades Comuns, do inglês *Common Weakness Enumeration*
DVWA	*Damn Vulnerable Web Application*
EPP	*Endpoint Protection Platforms*
GDPR	Regulamento Geral de Proteção de Dados, do inglês *General Data Protection Regulation*
GPU	*Graphics Processing Unit*
IA	Inteligência Artificial, do inglês *Artificial Intelligence*
IDS	*Intrusion Detection System*
IOC	Indicador de Comprometimento, do inglês *Indicator of Compromise*
IPS	*Intrusion Prevention System*
IR	*Intermediate Representation*
KQL	*Kusto Query Language*

LLM	Modelo de linguagem de grande escala, do inglês, *Large Language Model*
ML	Aprendizado de Máquina, do inglês *Machine Learning*
NIST	Instituto Nacional de Padrões e Tecnologia, do inglês *National Institute of Standards and Technology*
NVD	Banco Nacional de Vulnerabilidades, do inglês *National Vulnerability Database*
OSCTI	*Open-Source Cyber Threat Intelligence*
PCI-DSS	Padrão de Segurança de Dados da Indústria de Cartões de Pagamento, do inglês *Payment Card Industry Data Security Standard*
PLN	Processamento de Linguagem Natural, do inglês *Natural Language Processing*
QA	*Question Answering*
RAG	*Retrieval-Augmented Generation*, em português, Geração Aumentada por Recuperação
RNN	*Recurrent Neural Networks*, Redes Neurais Recorrentes.
SIEM	Sistema de Gerenciamento de Informações e Eventos de Segurança, do inglês *Security Information and Event Management*
SOC	Centro de Operações de Segurança, do inglês *Security Operations Center*
SPL	*Splunk Processing Language*
TF-IDF	Frequência de Termo–Frequência Inversa de Documento, do inglês *Term Frequency–Inverse Document Frequency*
TTP	*Tactics, Techniques, and Procedures* — em português, Táticas, Técnicas e Procedimentos
URL	*Uniform Resource Locators*
UUID	*Universally Unique Identifier*

SUMÁRIO

1 INTRODUÇÃO

1.1 Motivação

A segurança cibernética contemporânea sustenta a estabilidade de serviços essenciais, como saúde, energia, finanças e administração pública, enquanto enfrenta uma pressão crescente decorrente tanto do aumento no volume de ameaças quanto da complexidade das infraestruturas tecnológicas modernas. Os ataques cibernéticos alimentam um ciclo contínuo de exploração de vulnerabilidades, comprometendo desde a integridade de dados sensíveis até a continuidade operacional de organizações inteiras. Esse ciclo demanda abordagens de defesa cada vez mais ágeis e adaptativas, capazes de acompanhar a sofisticação dos ataques e a expansão de infraestruturas tecnológicas heterogêneas (Shah *et al.*, 2022).

Por infraestrutura heterogênea, entende-se um ambiente composto por diferentes tecnologias, dispositivos, sistemas operacionais, plataformas e arquiteturas que coexistem e interagem entre si. Tal heterogeneidade amplia a complexidade de gestão e de proteção, pois cada componente possui vulnerabilidades específicas e demanda estratégias distintas de defesa. Nesse contexto, um único ponto fraco pode comprometer todo o ecossistema, o que reforça a necessidade de uma defesa cibernética inovadora e adaptativa (Shah *et al.*, 2022; PALO ALTO NETWORKS, 2025; EXABEAM, 2025).

A dimensão desse desafio pode ser quantificada pelo ritmo de divulgação de vulnerabilidades. Só em 2025 foram publicados aproximadamente 48 mil registros de Vulnerabilidades e Exposições Comuns, do inglês *Common Vulnerabilities and Exposures* (CVE)s — uma média de cerca de 133 novas vulnerabilidades por dia e um crescimento de aproximadamente 21% em relação a 2024 (Gamblin, 2026). O volume tornou-se de tal forma insustentável que, em abril de 2026, o Instituto Nacional de Padrões e Tecnologia, do inglês *National Institute of Standards and Technology* (NIST) anunciou que seu Banco Nacional de Vulnerabilidades, do inglês *National Vulnerability Database* (NVD) deixaria de enriquecer todos os CVEs recebidos, passando a priorizar apenas uma fração deles, em razão de um aumento de 263% nas submissões entre 2020 e 2025 (National Institute of Standards and Technology, 2026). Este cenário torna evidente que o esforço manual de catalogação e análise de ameaças já não acompanha a velocidade com que elas surgem.

A urgência dessa adaptação se tornou ainda mais evidente diante da diminuição progressiva do intervalo entre a divulgação de uma vulnerabilidade e sua exploração. Os *zero-days*, ataques que exploram falhas ainda desconhecidas do desenvolvedor, sempre constituíram um caso particular em que a exploração precede a correção; o que se observa nos últimos anos, contudo, é a inversão da média desse intervalo. Relatórios de inteligência de ameaças documentam essa trajetória de forma quantitativa: o tempo médio para exploração (do inglês *time-to-exploit*) caiu de aproximadamente 63 dias em 2018 para cerca de 32 dias em 2023, cruzou o zero em 2024 e atingiu, em 2025, uma estimativa de -7 dias (Mandiant, 2026). Em

outras palavras, a exploração antecipando a divulgação deixou de ser exceção, característica dos *zero-days*, e passou a representar o comportamento médio do ecossistema de ameaças. A consequência estratégica é direta, pois quando a exploração precede o *patch* de correção, a aplicação das correções deixa de ser o controle decisivo, e a capacidade de detectar a atividade maliciosa em tempo hábil passa a ser o fator determinante para conter o ataque (Rapid7 Labs, 2026). A velocidade de detecção se tornou a linha de frente da defesa.

É precisamente neste contexto que se insere a engenharia de detecção, do inglês *detection engineering*, área da cibersegurança dedicada ao projeto, à validação e à manutenção sistemática das lógicas de detecção que alimentam os mecanismos de defesa. Os Sistema de Gerenciamento de Informações e Eventos de Segurança, do inglês *Security Information and Event Management* (SIEM)s, se consolidaram como pilares dessa atividade, correlacionando grandes volumes de registros automáticos de eventos (*logs*) gerados continuamente por sistemas, aplicações, redes e dispositivos. Assim como os *firewalls*, os *Intrusion Detection System* (IDS)s e os *Intrusion Prevention System* (IPS)s, os SIEMs operam fundamentalmente com base em regras de detecção. Tais regras, contudo, nem sempre acompanham a evolução das técnicas maliciosas, exigindo atualização constante (Shah *et al.*, 2022; Podzins; Romanovs, 2019).

O gargalo da engenharia de detecção, entretanto, não está apenas na defasagem das regras, mas no próprio esforço humano de produzi-las e mantê-las. Pesquisas recentes do setor apontam que a fadiga de alertas, do inglês *alert fatigue*, figura entre as principais preocupações dos Centro de Operações de Segurança, do inglês *Security Operations Center* (SOC)s, e que os falsos positivos constituem o desafio mais citado na detecção de ameaças, sobrecarregando equipes já afetadas pela escassez de profissionais qualificados (SANS Institute, 2025). Cada regra mal calibrada gera ruído; cada ruído consome o tempo de analistas que precisariam estar investigando ameaças reais. Reduzir o custo humano de produzir regras de detecção precisas e atualizadas é um problema substancial da defesa cibernética moderna.

Dentre as iniciativas que buscam padronizar e acelerar essa produção, destacam-se as regras *Sigma*: um formato aberto e descritivo, escrito em YAML (linguagem de serialização de dados legível e estruturada), voltado à definição de lógicas de detecção independentes de plataforma. Por serem convertíveis para linguagens nativas de diferentes SIEMs, como *Splunk Processing Language* (SPL) ou *Elasticsearch Query*, as regras *Sigma* favorecem a colaboração entre equipes e a portabilidade das detecções (SIGMAHQ, 2025a). No entanto, sua criação e manutenção manuais permanecem lentas e propensas a lacunas, sobretudo diante de ameaças emergentes como um *ransomware* (tipo de *malware* que sequestra dados ou sistemas e exige resgate) ou um *zero-day exploit*, ataque que explora uma falha de segurança ainda desconhecida do desenvolvedor e para a qual não há correção disponível (SIGMAHQ, 2025a; EXABEAM, 2025).

Diante deste conjunto de pressões — o volume insustentável de vulnerabilidades, a compressão do tempo de exploração e o custo humano da engenharia de detecção manual, a aplicação de técnicas de Inteligência Artificial, do inglês *Artificial Intelligence* (IA) na geração au-

tomatizada de regras apresenta-se como uma direção promissora, capaz de reduzir o intervalo entre a divulgação de uma ameaça e sua cobertura defensiva, preservando a qualidade técnica das regras produzidas. O estudo de Lempinen, Pyyny e Juntunen (2023) apud Hasanov *et al.* (2024) demonstrou que a integração entre Modelo de linguagem de grande escala, do inglês, *Large Language Model* (LLM), e o SIEM *Wazuh* [1] resultou em eficácia na análise de *logs*, na detecção de padrões de ameaças e na automação de respostas de segurança, beneficiando sobretudo usuários com conhecimento limitado na área. Tal evidência sugere que LLMs podem ser adaptados para apoiar a geração de regras *Sigma* a partir de dados de ameaças.

Sendo assim, este trabalho propõe a implementação de um agente autônomo capaz de automatizar a criação de regras *Sigma* a partir de referências externas de inteligência de ameaças, tipicamente CVEs, Indicador de Comprometimento, do inglês *Indicator of Compromise* (IOC)s, *logs* de incidentes e notícias de vulnerabilidades recém-divulgadas, acessadas durante o processo de geração. A proposta é concebida como ferramenta de apoio à engenharia de detecção, na qual o analista permanece como instância final de validação no fluxo operacional, ainda que a presente investigação se concentre na avaliação da geração automática propriamente dita. O agente combina recuperação aumentada por geração (*Retrieval-Augmented Generation*, em português, Geração Aumentada por Recuperação (RAG)), sobre um repositório consolidado de regras *Sigma*, consulta a fontes externas via busca na *web* e validação iterativa (sintática e semântica) das regras produzidas por um Modelo de Linguagem de Grande Escala. Ao final, os resultados obtidos por diferentes estratégias de geração, com complexidade arquitetural crescente, são comparados de modo a avaliar o ganho efetivo proporcionado pela arquitetura agêntica proposta.

1.2 Objetivos

1.2.1 Objetivo geral

Desenvolver um agente autônomo, de estrutura local, capaz de gerar regras *Sigma* de forma automatizada a partir de fontes diversificadas de inteligência de ameaças, com o propósito de acelerar a operacionalização de detecções em SIEMs.

1.2.2 Objetivos específicos

- Replicar regras *Sigma* a partir de LLMs comerciais.
- Gerar regras *Sigma* a partir do agente autônomo construído.

[1] O *Wazuh* é uma solução de segurança de código aberto com funcionalidades de *Host-based Intrusion Detection System* (HIDS) e SIEM (Wazuh, Inc., 2024).

- Desenvolver um agente autônomo para a geração de regras *Sigma*.
- Comparar os resultados dos métodos utilizados.
- Comparar métricas dos metadados derivados do agente.

1.3 Estrutura do trabalho

Este trabalho possui seis capítulos, organizados de forma a conduzir o leitor do embasamento teórico à abordagem metodológica que será adotada. No primeiro capítulo, apresenta-se a introdução do tema, defendendo a motivação, os objetivos e a justificativa do estudo. O capítulo 2 disserta sobre conceitos necessários para se compreender o assunto do trabalho: sistemas SIEM, regras *Sigma* e LLMs — o que são, como se configuram na segurança da informação e como se relacionam entre si diante da mitigação de ameaças. Já o capítulo 3 discute trabalhos relacionados, oferecendo uma visão crítica sobre pesquisas e soluções correlatas. O capítulo 4 descreve os materiais e os métodos utilizados, detalhando as etapas do desenvolvimento da proposta e quais serão as métricas de avaliação. No capítulo 5 estão os produtos do trabalho proposto, os resultados obtidos, a interpretação dos mesmos e discussão sobre as limitações encontradas. No último capítulo, a conclusão e sugestão de trabalhos futuros.

2 REFERENCIAL TEÓRICO

Este capítulo apresenta os fundamentos teóricos que sustentam o desenvolvimento do agente proposto. São discutidos os conceitos de SOC e SIEM na área de segurança da informação, as bases de conhecimento sobre vulnerabilidades e técnicas adversárias, o padrão *Sigma* para regras de detecção, os modelos de linguagem de grande escala (LLMs), a geração aumentada por recuperação (RAG) e os agentes autônomos como paradigma de orquestração de LLMs.

2.1 Segurança da Informação e SOC

Em organizações de médio e grande porte a defesa cibernética não costuma ser tarefa de uma única ferramenta. O volume e a heterogeneidade dos eventos gerados por servidores, *firewalls*, sistemas de detecção de intrusão (IDS), aplicações em nuvem e dispositivos de usuários tornam inviável a inspeção manual contínua, exigindo a composição de equipes e infraestruturas dedicadas ao monitoramento, à detecção e à resposta a incidentes (Vielberth *et al.*, 2020; Shah *et al.*, 2022). É neste contexto que surgem os SOC, unidades especializadas que centralizam pessoas, processos e tecnologias para sustentar a postura defensiva de uma organização em tempo integral.

Segundo Vielberth *et al.* (2020), um SOC bem estruturado articula três pilares interdependentes: analistas humanos com diferentes níveis de experiência, procedimentos formalizados de triagem e resposta e uma camada tecnológica capaz de coletar, correlacionar e priorizar evidências de ameaças. A operação típica se organiza em níveis funcionais, em que no *Tier* 1 os analistas iniciais executam a triagem dos alertas, no *Tier* 2 os investigadores intermediários conduzem a análise de incidentes e no *Tier* 3 os especialistas de nível mais alto realizam e perícia forense (Tariq *et al.*, 2025). O componente tecnológico central que sustenta essa cadeia é o sistema SIEM, descrito na próxima seção.

2.1.1 SIEM

Um SIEM atua como o núcleo de inteligência operacional em ambientes de cibersegurança, agregando e correlacionando *logs* — registros automáticos de eventos gerados por sistemas, aplicações, redes e dispositivos, provenientes de toda a infraestrutura de Tecnologia da Informação. Como já citado anteriormente, as fontes típicas incluem *firewalls*, sistemas de detecção e prevenção de intrusão (IDS/IPS), servidores de aplicação e de banco de dados, controladores de domínio, *endpoints* e serviços em nuvem (González-Granadillo; González-Zarzosa; Diaz, 2021). Seu propósito reside na capacidade de correlacionar eventos aparentemente desconexos e identificar padrões que ferramentas isoladas não o fazem.

Convém distinguir o SIEM de outras soluções pontuais de defesa, como antivírus, *firewalls* ou plataformas de proteção de *endpoint* (EPP). Tais soluções operam de forma localizada, protegendo um *host*, um perímetro de rede ou um vetor específico, e geram alertas específicos de seu escopo. O SIEM, por outro lado, é uma camada agregadora: ele consome os *logs* produzidos por essas ferramentas, normaliza formatos heterogêneos para um esquema comum e aplica regras de correlação capazes de identificar, na conjunção de múltiplos eventos, padrões de ataque que permaneceriam imperceptíveis em uma análise isolada (González-Granadillo; González-Zarzosa; Diaz, 2021; Sheeraz *et al.*, 2024). Conforme destacam Podzins e Romanovs (2019), essa contextualização é decisiva em ações automatizadas contra ameaças complexas, como ataques *zero-day*, em que, por exemplo, a correlação entre uma tentativa de força bruta no *Active Directory* (serviço de diretório da *Microsoft* responsável pelo gerenciamento de identidades, recursos e permissões da rede interna) e um tráfego suspeito no *firewall* pode disparar uma resposta imediata.

Nos SOCs, o SIEM assume papel operacional indispensável: consolida alertas de diferentes fontes e formatos, prioriza incidentes com base na criticidade e orquestra fluxos de resposta (*workflows*). Sem essa camada, os analistas perdem a capacidade de detectar fases avançadas de ataque, como movimento lateral e escalação de privilégios — etapas em que o invasor se desloca pelos sistemas do alvo acumulando privilégios até atingir o nível administrativo (Podzins; Romanovs, 2019). A ausência desse tipo de visão integrada está associada a violações de longa duração, como foi o caso da rede mundial de hotéis *Starwood*, em que a intrusão permaneceu não identificada por aproximadamente quatro anos e resultou no vazamento de dados de cerca de 383 milhões de hóspedes ao redor do mundo (Shepardson, 2019). Essa visão unificada transforma o SIEM no "sistema nervoso"do SOC, permitindo que analistas investiguem ameaças sistêmicas e não apenas alertas isolados.

A eficácia de um SOC, porém, está intrinsecamente ligada à robustez do SIEM e à calibração das suas regras de detecção. Regras mal calibradas geram excesso de falsos positivos e instâncias não otimizadas podem produzir mais de mil alertas diários, sobrecarregando as equipes. Segundo Podzins e Romanovs (2019), um analista isolado consegue investigar com profundidade entre 5 e 15 alertas por dia, dependendo da criticidade. Esse desequilíbrio configura o fenômeno conhecido como *alert fatigue*, ou fadiga de alertas: a exposição contínua a grandes volumes de notificações de baixo valor leva os analistas à dessensibilização, ao prolongamento do tempo de resposta e, em última instância, ao *burnout* profissional, com impactos diretos na qualidade da defesa (Tariq *et al.*, 2025). A literatura recente identifica a fadiga de alertas como um dos principais desafios contemporâneos dos SOCs e, justamente por isso, motiva a busca por mecanismos automatizados de triagem, correlação e geração de regras mais precisas — preocupação central do presente trabalho.

A importância estratégica do SIEM também se ancora na sua capacidade de reduzir o tempo de detecção de ameaças. Podzins e Romanovs (2019) citam o relatório de violações da *Verizon* de 2018, que apontava cerca de 68% das violações sendo descobertas meses ou

mais após o ataque inicial (Business, 2018); já o relatório de 2025 informa que o tempo médio de remediação de vulnerabilidades em dispositivos de borda (especialmente falhas de *zero-day*) é de 32 dias (Business, 2025). Essa lacuna pode ser substancialmente reduzida com um SIEM bem implementado, em virtude da correlação de eventos em tempo real. Adicionalmente, a plataforma SIEM serve de alicerce para a evolução tecnológica do SOC: é sobre ela que se integram componentes mais recentes, como módulos de inteligência artificial e aprendizado de máquina, que ampliam a detecção de ameaças sutis e habilitam a automação de respostas (Tendikov *et al.*, 2024).

Do ponto de vista econômico, embora o SIEM envolva custos elevados, com licenças baseadas em volume de *logs*, infraestrutura dedicada e equipe especializada em regime ininterrupto, o investimento se justifica diante da magnitude das perdas associadas a incidentes cibernéticos. Em 2019, a *Accenture* projetava perdas globais da ordem de US$ 5,2 trilhões para os 5 anos seguintes (Accenture, 2019 apud Podzins; Romanovs, 2019), tendência que se confirmou nos anos posteriores: o relatório anual da IBM de 2024 ((IBM Security, 2024)) registrou um custo médio global de US$ 4,88 milhões por violação de dados, com aumento de 10% em relação ao ano anterior. O último, de 2025, no entanto, trouxe uma pequena reviravolta: o custo médio global caiu 9% em relação a 2024, motivado por melhorias na identificação e contenção de incidentes (IBM Security, 2025). Para organizações com dados sensíveis ou sujeitas a regulações como o Regulamento Geral de Proteção de Dados, do inglês *General Data Protection Regulation* (GDPR), da União Europeia, ou o PCI-DSS, norma globalmente reconhecida para a segurança de dados de cartões de pagamento, o SIEM deixa de ser uma opção e torna-se requisito operacional. Em síntese, o SIEM mantém-se como pilar irrevogável da segurança corporativa, não apenas pela sua tecnologia isolada, mas por construir o ecossistema integrado de detecção e resposta sobre o qual se constroem as defesas modernas.

2.1.2 SIEMs Inteligentes

Apesar de sua centralidade operacional, o sistema SIEM tradicional opera majoritariamente sobre regras estáticas — definições escritas por especialistas que descrevem condições de alertas a partir de padrões conhecidos. Essa abordagem apresenta duas limitações sistemáticas: a dificuldade em detectar ameaças cuja assinatura ainda não foi formalizada, como ataques *zero-day*, e a alta taxa de falsos positivos quando as regras não acompanham a dinâmica do ambiente protegido (González-Granadillo; González-Zarzosa; Diaz, 2021; Nurusheva *et al.*, 2024). González-Granadillo, González-Zarzosa e Diaz (2021) observam, especificamente, que pouquíssimas soluções de SIEM dispõem de mecanismos de correlação avançada capazes de realizar uma análise histórica e de desvios necessária, por exemplo, para a detecção posterior do ataques *zero-day*. Portanto, é a partir da integração de técnicas de Aprendizado de Máquina, do inglês *Machine Learning* (ML) e IA, buscando superar tais limitações, que se originam os chamados SIEMs inteligentes.

De acordo com Saddi *et al.* (2024), a IA generativa é capaz de produzir dados sintéticos e simular cenários de ataque, replicando, por exemplo, a atuação de um invasor ou a propagação de *malware*, o que permite tanto treinar modelos de detecção quanto avaliar defesas antes de sua implantação, sustentando uma postura proativa de identificação de ameaças emergentes. Essa capacidade é amplificada por algoritmos de ML que analisam padrões contextuais em grandes volumes de *logs*, correlacionam eventos aparentemente desconexos, como por exemplo, tráfego anômalo combinado com tentativas de força bruta, e identificam comportamentos compatíveis com técnicas adversárias documentadas. Tendikov *et al.* (2024) reforçam que a integração de IA e ML em SIEMs aprimora a correlação de grandes volumes de eventos em tempo real e permite que esses sistemas aprendam a partir de dados históricos e se adaptem dinamicamente a ameaças emergentes. Esse aprimoramento incide diretamente sobre a fadiga dos analistas discutida na seção anterior: Tariq *et al.* (2025) documentam que mecanismos de automação e triagem assistida por ML reduzem a carga cognitiva associada ao alto volume de alertas e ao excesso de falsos positivos, principais causas do esgotamento nos SOCs.

Nurusheva *et al.* (2024) demonstram que algoritmos de detecção de anomalias, análise preditiva e limiares adaptativos permitem identificar atividades sutis que escapariam das regras estáticas ou à análise manual, e propõem um sistema autoadaptativo, capaz de identificar ameaças inéditas em tempo real sem depender de regras predefinidas. Um aspecto central desse paradigma é a possibilidade de reduzir a dependência de bases previamente rotuladas [1]: como destacado por Tariq *et al.* (2025), algoritmos não supervisionados conseguem modelar o comportamento típico de uma rede e detectar anomalias sem a necessidade de conjuntos de dados rotulados, o que os torna particularmente eficazes na identificação de ameaças novas, embora ao custo de uma taxa de falsos positivos superior à dos métodos supervisionados. Do ponto de vista conceitual, esse comportamento decorre das próprias características do aprendizado de máquina, cuja capacidade de aprender padrões a partir de dados, sem programação explícita, fundamenta tanto as abordagens supervisionadas quanto as não-supervisionadas (Jordan; Mitchell, 2015). Ao combinar esses dois tipos de modelo, o SIEM com ML ajusta-se continuamente às mudanças no ambiente e antecipa novos vetores de ataque (Sheeraz *et al.*, 2024).

Por fim, compreende-se que depender exclusivamente de analistas humanos é inviável diante do crescimento exponencial dos dados e da sofisticação das ameaças. A incorporação de IA e ML em SIEMs representa, portanto, um avanço significativo da cibersegurança, conferindo precisão, adaptabilidade e automação a um cenário cada vez mais dinâmico e hostil. Esse movimento, no entanto, ainda enfrenta desafios relevantes, sobretudo no que tange à interpretabilidade dos modelos, à dependência de dados de qualidade para o treinamento e à necessidade de regras formais que estabeleçam o vocabulário comum entre as diferentes ferramentas. Seguindo este último ponto, nas próximas seções se discutirá sobre a efetividade

[1] Uma base rotulada (do inglês *labeled dataset*) condiz a uma coleção de pares entrada-saída, onde cada amostra de entrada (por exemplo, linha de *log* ou fluxo de rede) recebeu uma anotação confiável sobre o que se é de fato

de qualquer SIEM; sendo inteligente ou não, depende de regras de detecção formalizadas em padrões abertos e interoperáveis, ancoradas em bases consolidadas de conhecimento sobre vulnerabilidades e técnicas adversárias.

2.2 Bases de Conhecimento sobre Vulnerabilidades e Técnicas Adversárias

A construção de regras de detecção em SIEMss depende de um vocabulário comum para descrever vulnerabilidades, fraquezas e comportamentos adversários. Esse vocabulário é fornecido por bases abertas, mantidas por organizações de pesquisa e agências governamentais, que catalogam de forma sistemática os elementos de interesse para a defesa cibernética. Sem esses padrões, regras escritas por diferentes equipes seriam incomparáveis, e a interoperabilidade entre ferramentas seria inviável (Roy *et al.*, 2023). O agente proposto neste trabalho consulta quatro dessas bases ao longo de sua execução: CVE, CWE, NVD e MITRE ATT&CK. As subseções a seguir apresentam cada uma delas e justificam seu papel na geração das regras *Sigma*.

2.2.1 CVE e CWE

O CVE é um catálogo público de vulnerabilidades, operado pelo MITRE desde 1999 sob patrocínio da Agência de Segurança de Infraestrutura e Cibernética, do inglês *Cybersecurity and Infrastructure Security Agency* (CISA) — agência norte-americana de segurança cibernética (The MITRE Corporation, 2024)). Cada vulnerabilidade divulgada recebe um identificador único no formato CVE-AAAA-NNNN, em que AAAA é o ano da publicação e NNNN um número sequencial. Em 2024, o catálogo já ultrapassava 250 mil registros (The MITRE Corporation, 2025b). A principal função do CVE é eliminar ambiguidade: quando duas ferramentas mencionam "CVE-2021-44228", há certeza de que ambas se referem à mesma falha — aqui, a vulnerabilidade *Log4Shell*, na biblioteca *Apache Log4j*. Neste trabalho, o CVE atua como uma das entradas possíveis para o agente: quando o usuário fornece um identificador, ou quando a regra parte de uma descrição textual que o menciona, o agente consulta o catálogo para extrair contexto descritivo.

O Enumeração de Vulnerabilidades Comuns, do inglês *Common Weakness Enumeration* (CWE), também mantido pelo MITRE, opera em outro nível de abstração. Enquanto o CVE cataloga vulnerabilidades específicas em produtos identificáveis, o CWE classifica as categorias gerais de fraqueza subjacentes, como *buffer overflow*, falhas de validação de entrada, exposição de dados sensíveis e centenas de outras. Uma vulnerabilidade CVE costuma estar associada a um ou mais CWEs que descrevem a natureza do problema (The MITRE Corporation, 2025a).

Para a geração de regras, essa abstração é valiosa: padrões CWE generalizam-se para famílias inteiras de vulnerabilidades, incluindo variantes ainda não catalogadas. O agente pro-

posto obtém os CWEs associados a um CVE por meio da consulta ao NVD, descrita na subseção 4.2.3.4.

2.2.2 MITRE ATT&CK

Em complemento aos catálogos de vulnerabilidades e fraquezas, o MITRE ATT&CK (*Adversarial Tactics, Techniques, and Common Knowledge*) fornece uma taxonomia de *Tactics, Techniques, and Procedures* — em português, Táticas, Técnicas e Procedimentos (TTP) efetivamente utilizados por adversários em incidentes reais (Strom *et al.*, 2018). A diferença é importante: CVE e CWE descrevem o que pode ser explorado; o ATT&CK descreve como os atacantes agem ao longo das fases de uma intrusão, desde o reconhecimento inicial até a exfiltração de dados.

A matriz *Enterprise*, variante mais utilizada, organiza essas TTPs em 14 táticas e centenas de técnicas, cada uma identificada por um código, como por exemplo, T1190 para *Exploit Public-Facing Application*. Em uma revisão sistemática, Roy *et al.* (2023) identificaram mais de 50 contribuições científicas que utilizam o ATT&CK como base para pesquisas em inteligência de ameaças, detecção de intrusão e resposta a incidentes, destacando-o como referência consolidada na academia e na indústria.

No agente proposto, o ATT&CK é utilizado para validar as *tags* de técnicas que o LLM inclui na regra *Sigma* gerada, visto que modelos de linguagem podem inventar identificadores inexistentes (por exemplo, *attack.exploitation* em vez do real *attack.initial-access*); a consulta à base oficial do ATT&CK permite remover ou sinalizar referências inválidas antes da entrega final da regra.

Cabe registrar que o MITRE mantém outras bases relacionadas, em particular o Common Attack Pattern Enumerations and Classifications (CAPEC), que cataloga padrões de ataque em um nível de abstração mais alto que o ATT&CK. Este catálogo não foi incorporado ao agente neste trabalho porque sua granularidade não corresponde ao formato das regras *Sigma*, que operam sobre técnicas concretas e mapeáveis a eventos de *log* — exatamente o nível em que o ATT&CK atua.

2.2.3 NVD e CVSS

O NVD é o repositório oficial do governo norte-americano para vulnerabilidades de segurança, mantido pelo NIST, e atua como camada de enriquecimento do CVE. A cada vulnerabilidade identificada, o NVD acrescenta metadados estruturados como produtos e versões afetados, mapeamentos para CWE e pontuações de severidade calculadas segundo o Sistema de Pontuação de Vulnerabilidades Comuns, do inglês *Common Vulnerability Scoring System* (CVSS) (National Institute of Standards and Technology, 2025; Booth; Rike; Witte, 2013). O

CVSS, por sua vez, é um sistema híbrido; produz uma pontuação numérica de 0,0 a 10,0 a partir de métricas como vetor de ataque, complexidade, privilégios requeridos e impacto sobre confidencialidade, integridade e disponibilidade, e mapeia essa pontuação em categorias qualitativas (*None, Low, Medium, High e Critical*), oferecendo simultaneamente critério quantitativo para ordenação e classificação intuitiva para consumo humano (Forum of Incident Response and Security Teams, 2019). No agente proposto, o NVD faz uma requisição para obter a descrição da vulnerabilidade, os CWEs associados e a pontuação CVSS — dados que alimentam tanto o conteúdo descritivo da regra quanto a calibração do campo *level* (que indica a criticidade da regra *Sigma*).

2.3 Regras *Sigma*

Sigma (*Generic Signature Format for SIEM Systems*, “Assinatura em formato genérico para sistemas SIEM” em tradução livre (SIGMAHQ, 2025b)) é o projeto de código aberto de um padrão genérico de descrição de *logs* de eventos, aplicável a qualquer tipo de *log*, com o intuito de compartilhar a lógica de detecção de ameaças ou anomalias entre diferentes SIEMs. Ao operar com dados de ameaças de múltiplas fontes, o *Sigma* capacita os caçadores de ameaças a correlacionar eventos que passariam despercebidos por abordagens convencionais. Dessa forma, se obtém conhecimentos estratégicos sobre dados confiáveis e antecipa-se tentativas de ataque em estágios iniciais desta cadeia de eventos (McGowan, 2025).

Os mantenedores da iniciativa consideram que excelentes análises podem ser encontradas na *internet*, incluindo IOCs e regras YARA[2] para detectar arquivos ou ferramentas maliciosos, por exemplo, mas que muitas vezes limitam-se a descrições específicas, funcionais para poucos *softwares*. Portanto, da necessidade de haver um método de descrição específico ou genérico de eventos de *logs*, com portabilidade para vários SIEMs, a fim de aprimorar os recursos de detecção para todos, surgiu o Sigma (SIGMAHQ, 2025b).

O projeto de código aberto foi criado em 2017 por Florian Roth e Thomas Patzke (SIGMAHQ, 2025a), inspirados em padrões já existentes nas áreas de análise de *malware* e monitoramento de rede. Como já dito, o padrão *Sigma* se diferencia por lidar com vários tipos de esquemas de *logs*, unindo padrões que já existem e se diferenciam entre si (McGowan, 2025).

O projeto carrega essa denominação pois se considera um ecossistema. Envolve três componentes: o formato *Sigma*, a ferramenta *Sigma* e a coleção de regras *Sigma*. Este último diz respeito ao repositório no *Github*, que possui mais de 3 mil regras de detecção de diferentes tipos; por ser de código aberto, pode receber sugestões de qualquer pessoa disposta a contribuir (SIGMAHQ, 2025b). Já a ferramenta *Sigma*, refere-se a um conversor de regras disponível para *download* — o *Sigma Command Line Interface* (CLI). Com ele é possível converter as regras *Sigma* para as regras usadas no SIEM de interesse do usuário; *Elasticsearch*, *CrowdStrike*,

[2] Uma regra YARA é um arquivo de extensão “.yar” contido de expressões lógicas usadas para classificar e identificar amostras de *malware* (PICUS SECURITY, 2025).

IBM QRadar AQL e *Splunk* são algumas marcas registradas para as quais essa integração está disponível.

2.3.1 Sigma e a mitigação de ameaças

As regras *Sigma* formam um conjunto estruturado de princípios e métodos essenciais para o sucesso da investigação focada em determinar as ameaças que podem prejudicar um ou mais sistemas (*Threat Hunting*), oferecendo um padrão uniforme que orienta os profissionais de segurança na elaboração de regras criteriosas para detecção de ameaças ou anomalias. Segundo Dimitrov e Nikolov (2025), essas regras otimizam a identificação, avaliação e a resposta a incidentes, proporcionando decisões mais precisas e oportunas, além de uma base sólida para controlar vetores de ameaças, minimizar riscos e prevenir violações de dados. Dessa forma, as regras *Sigma* constituem o alicerce de uma estratégia de segurança proativa e resiliente, capaz de antecipar e mitigar possíveis ataques de forma sustentável.

A caça a ameaças é um processo defensivo proativo que visa descobrir e neutralizar invasores, mapeando vulnerabilidades e antecipando movimentos adversos por meio da análise contínua de comportamentos anômalos e incidentes em curso. Nesse cenário, as regras *Sigma*, desde sua padronização, em 2017, oferecem uma linguagem comum para correlação de eventos em múltiplas plataformas SIEM, unificando dados de sistemas heterogêneos e acelerando a detecção de padrões suspeitos. Conforme demonstrado por Dimitrov e Nikolov (2025), essa unificação reduz significativamente a carga de trabalho manual dos analistas, permitindo identificar ameaças ainda na fase inicial de comprometimento, elevando a eficácia das operações de *Threat Hunting*, ou “caça a ameaças”.

No entanto, ainda que a aplicação das regras *Sigma* viabilize inúmeros cenários benquistos, vale ressaltar que, ainda assim, os SIEMs continuarão sendo heterogêneos, pois utilizam formatos e padrões de dados diferentes, situação inclusive de eventos de diferentes dispositivos. As regras *Sigma* surgem como uma forma de expressar a regra ou a ameaça e o que deve ser buscado, mas ainda precisa ser convertida para o SIEM de destino (através do CLI), que já possui suas próprias regras.

2.3.2 Estrutura

As regras *Sigma* são arquivos YAML contidos de informações que viabilizam a detecção de comportamento estranho ou malicioso quando da inspeção de arquivos de *logs* dentro do contexto de um SIEM (SIGMAHQ, 2025c).

A compreensão da estrutura de uma regra *Sigma* e como é usada para detecção faz-se importante em razão de que parte da metodologia envolverá selecionar as informações mais críticas das regras e avaliá-las.

As regras são divididas em três grupos principais de instruções (SIGMAHQ, 2025c):

- Detecção (*detection*): qual comportamento malicioso a regra está procurando;
- Fonte do log (*logsource*): quais tipos de logs a detecção deve procurar; e
- Metadados (*title*, *id*, *status*, etc.): detalhes da regra.

A Figura 1 é um exemplo de regra *Sigma* que está disponível na página oficial do projeto.

Figura 1 – Exemplo de regra *Sigma*

```yaml
# ./rules/cloud/okta/okta_user_account_locked_out.yml
title: Okta User Account Locked Out
id: 14701da0-4b0f-4ee6-9c95-2ffb4e73bb9a
status: test
description: Detects when a user account is locked out.
references:
    - https://developer.okta.com/docs/reference/api/system-log/
    - https://developer.okta.com/docs/reference/api/event-types/
author: Austin Songer @austinsonger
date: 2021-09-12
modified: 2022-10-09
tags:
    - attack.impact
logsource:
    product: okta
    service: okta
detection:
    selection:
        displaymessage: Max sign in attempts exceeded
    condition: selection
falsepositives:
    - Unknown
level: medium
```

Fonte: Captura de tela da página *Sigma Rules* em SigmaHQ (2025c).

A seção de detecção é a parte central da regra *Sigma*, pois especifica o alvo da regra no vasto volume de *logs*. Adiante neste trabalho este componente será mencionado como um dos principais para a análise da qualidade das regras geradas pelas LLMs. Vale mencionar que, dentro da seção de detecção, existem instruções para a escrita, assim como qualquer linguagem de programação; no entanto, num primeiro momento, basta compreender o esqueleto principal. Sendo assim, a segunda peça mais importante da regra é a fonte dos *logs*, o campo *logsource*; nele se especifica qual o tipo de *log* deve ser canalizado pela regra, ao invés de varrer todos os *logs* do SIEM. Por fim, as demais informações, os metadados, servem para guiar o usuário sobre o assunto da regra (SIGMAHQ, 2025c).

Para fins de orientação quando da criação de uma nova regra, é disponibilizado um esqueleto da regra com instruções (Figura 2) no repositório *Sigma* (Roth, 2022).

Figura 2 – Exemplo de regra *Sigma*

```
title: a short capitalised title with less than 50 characters
id: generate one here https://www.uuidgenerator.net/version4
status: experimental
description: A description of what your rule is meant to detect
references:
    - A list of all references that can help a reader or analyst understand the meaning of a triggered rule
tags:
    - attack.execution  # example MITRE ATT&CK category
    - attack.t1059      # example MITRE ATT&CK technique id
    - car.2014-04-003   # example CAR id
author: Michael Haag, Florian Roth, Markus Neis  # example, a list of authors
date: 2018/04/06  # Rule date
logsource:                      # important for the field mapping in predefined or your additional config files
    category: process_creation  # In this example we choose the category 'process_creation'
    product: windows            # the respective product
detection:
    selection:
        FieldName: 'StringValue'
        FieldName: IntegerValue
        FieldName|modifier: 'Value'
    condition: selection
fields:
    - fields in the log source that are important to investigate further
falsepositives:
    - describe possible false positive conditions to help the analysts in their investigation
level: one of five levels (informational, low, medium, high, critical)
```

Fonte: Captura de tela do código genérico na página *Rule Creation Guide* no repositório Sigma.

2.4 Processamento de Linguagem Natural

O Processamento de Linguagem Natural, do inglês *Natural Language Processing* (PLN) é a área da computação que busca fazer com que máquinas compreendam e produzam a linguagem humana. Segundo Liddy (2001), trata-se de uma abordagem teórica e tecnologicamente fundamentada para analisar textos, com o objetivo de alcançar um processamento de linguagem semelhante ao humano. Para isso, o PLN permite ao computador analisar um texto sistematicamente em sete níveis de representação: fonológico (padrões de entonação e fonemas), morfológico (estrutura das palavras), léxico (significado individual das palavras), sintático (estrutura gramatical das frases), semântico (significado literal das frases), discursivo (coesão e estrutura dos textos) e pragmático (contexto) (Liddy, 2001; Jurafsky; Martin, 2009). Ao longo do tempo, as técnicas evoluíram dos sistemas simbólicos baseados em regras manuais para abordagens estatísticas e, mais recentemente, conexionistas (inspiradas no funcionamento do cérebro humano), impulsionadas pelo aumento do poder computacional e pela disponibilidade de grandes coleções de texto (Liddy, 2001; Chowdhary, 2020).

Sobre essa base, o PLN passou a sustentar aplicações que se tornaram centrais na computação contemporânea. A Recuperação de Informação (RI) encontra, em uma grande coleção de documentos, aqueles mais relevantes a uma consulta, ordenando-os por similaridade semântica em vez de apenas casar palavras idênticas (Manning; Raghavan; Schütze, 2008). A Extração de Informação (EI) parte de texto livre e deste são retirados dados estruturados, como entidades, datas, relações, igualmente quando um anúncio de emprego em texto corrido é convertido em campos de uma tabela (Chowdhary, 2020). A Tradução Automática (TA) converte um texto entre idiomas preservando o significado original, mantendo uma correspondência pró-

xima entre origem e destino. Já a Sumarização condensa documentos longos em versões mais curtas que retêm o essencial, podendo ser extrativa (seleciona frases do original) ou abstrativa (reescreve em novas palavras) (Jurafsky; Martin, 2009).

Os Sistemas de Pergunta-Resposta (QA) ocupam um nível distinto: em vez de transformar diretamente o texto de entrada, interpretam uma pergunta em linguagem natural e produzem uma resposta direta, frequentemente combinando RI, EI e sumarização como etapas internas (Jurafsky; Martin, 2009). A diferença essencial é o tipo de saída entregue ao usuário: enquanto a RI devolve uma lista de documentos para que ele mesmo leia e conclua, o QA devolve a conclusão já formulada. Todas essas aplicações enfrentam dificuldades comuns relacionadas à integração de conhecimento de mundo e ao raciocínio de senso comum, necessários para resolver ambiguidades contextuais (Chowdhary, 2020). Esse cenário começou a mudar no final da década de 2010, com o surgimento dos modelos de linguagem baseados em redes neurais profundas, em especial os Modelos de Linguagem de Grande Escala, descritos na próxima seção, cuja escala de dados e parâmetros permitiu superar boa parte dessas limitações sem depender de regras escritas à mão (Hasanov *et al.*, 2024).

Várias dessas aplicações reaparecem, de forma combinada, no agente desenvolvido neste trabalho. A recuperação de informação fundamenta a busca por regras Sigma semelhantes em uma base vetorial, núcleo da estratégia de geração aumentada por recuperação descrita na Seção 2.5. A extração de informação atua na identificação de indicadores técnicos, como identificadores CVE e *hashes*, a partir de texto livre. E a sumarização orienta a condensação do material coletado de fontes externas em uma descrição curta e factual da ameaça. Como é mostrado nas subseções seguintes, é justamente a possibilidade de unir essas tarefas em um único sistema, por meio de modelos de linguagem de grande escala, que torna essa arquitetura viável.

2.4.1 Modelos de Linguagem de Grande Escala (LLMs)

Os Modelos de Linguagem de Grande Escala (*Large Language Models*, LLMs) são sistemas de inteligência artificial treinados com bilhões ou trilhões de parâmetros, capazes de compreender e gerar textos em linguagem natural, ou seja, em formato de comunicação humano (Hasanov *et al.*, 2024). Sua arquitetura é baseada em *Transformers*: redes neurais que empregam mecanismos de *self-attention* (autoatenção) para focar simultaneamente em diferentes partes do texto, capturando relações contextuais de longo alcance entre as palavras (Vaswani *et al.*, 2017). Esses modelos aprendem por meio de aprendizado auto-supervisionado (*self-supervised learning*), estratégia em que o próprio texto fornece o sinal de treinamento: dada uma sequência incompleta, o modelo tenta prever o elemento ausente, extraindo padrões linguísticos a partir de imensos *corpora* [3] textuais sem rótulos explícitos (Hasanov *et al.*, 2024).

[3] *Corpora* é plural de *corpus* — termo em latim para "corpos" e no contexto de PLN significa coleções extensas e estruturadas de textos autênticos.

O mecanismo operacional central de um LLM é a predição sequencial de *tokens*, as menores unidades de texto em que o modelo divide sua entrada. Dada uma sequência de entrada, o modelo estima a probabilidade do próximo token, e essa habilidade é refinada continuamente ao longo do pré-treinamento. Ao operar sobre janelas de contexto de milhares de *tokens*, o modelo mantém coerência semântica em textos longos e responde a instruções formuladas em linguagem natural (Brown *et al*., 2020; Jurafsky; Martin, 2009).

Uma propriedade particularmente relevante para o presente trabalho é a capacidade dos LLMs de produzir saídas em formatos estruturados, como YAML, JSON ou código-fonte. É essa capacidade que permite ao modelo gerar regras *Sigma*, que são justamente "artefatos" estruturados em YAML, com campos formalmente definidos, e a arquitetura que viabilizou esse comportamento é o *Transformer*, descrita a seguir, enquanto as estratégias para orientar o modelo sem retreinamento são objeto da subseção 2.4.3.

2.4.2 Arquitetura *Transformer*

A arquitetura *Transformer*, proposta por Vaswani *et al.* (2017) no artigo "*Attention is All You Need*", é o substrato técnico que viabilizou os modelos de linguagem contemporâneos. Antes dela, o processamento de texto era dominado por redes neurais recorrentes (RNN), que processavam o texto palavra por palavra, em sequência, o que dificultava o paralelismo computacional e limitava a captura de dependências entre elementos distantes.

O *Transformer* substitui a recorrência por um mecanismo denominado *self-attention* (autoatenção). Em vez de processar um *token* de cada vez, o modelo analisa todos os *tokens* da entrada simultaneamente e calcula, para cada um, o quanto os demais *tokens* da sequência são relevantes para sua representação. O resultado é uma representação contextualizada de cada *token*, que incorpora informação de toda a sequência (Vaswani *et al*., 2017).

Essa formulação trouxe duas consequências práticas que explicam por que o *Transformer* se tornou a base dos LLMs. A primeira é o paralelismo: como todas as relações são calculadas simultaneamente, o treinamento torna-se massivamente paralelizável, viabilizando modelos com bilhões de parâmetros. A segunda é a captura de dependências de longo alcance: a relação entre dois tokens distantes é calculada com o mesmo custo da relação entre tokens adjacentes, eliminando o viés que penalizava informações distantes nas RNNs (Vaswani *et al*., 2017). A arquitetura original combina um encoder, voltado à compreensão da entrada, e um decoder, voltado à geração da saída; os LLMs empregados neste trabalho derivam de sua porção decoder, especializada na geração de texto.

2.4.3 Engenharia de *Prompt*

Diferentemente dos modelos clássicos de aprendizado de máquina, que precisam ser retreinados para cada nova tarefa, os LLMs podem ser condicionados em tempo de inferência por meio de instruções em linguagem natural, os *prompts*. A formulação dessas instruções constitui o ramo da "engenharia de *prompt*", cujos paradigmas mais estabelecidos foram caracterizados por Brown *et al.* (2020).

O primeiro paradigma é o *zero-shot*, em que o modelo recebe apenas a descrição da tarefa e a entrada, sem nenhum exemplo de saída esperada. Instruir o modelo a “gerar uma regra *Sigma* para detectar a vulnerabilidade CVE-2021-44228” é uma chamada *zero-shot*: o modelo precisa inferir, a partir do que aprendeu no pré-treinamento, o formato esperado e os detalhes a incluir. O segundo paradigma é o *few-shot*, em que o *prompt* inclui um pequeno número de exemplos (tipicamente entre dois e dez) que ilustram a transformação desejada entre entrada e saída. Esses exemplos atuam como demonstração contextual: o modelo identifica o padrão a partir das amostras e o reproduz, sem que qualquer parâmetro seja atualizado (Brown *et al.*, 2020).

A diferença entre os dois tem implicações práticas. O *zero-shot* depende inteiramente do conhecimento incorporado no pré-treinamento, sendo mais sensível a domínios pouco representados nos dados de origem; o *few-shot* fornece especificação adicional sem custo de treinamento, mas consome parte do orçamento de *tokens* da janela de contexto. Brown *et al.* (2020) demonstraram que o ganho do *few-shot* sobre o *zero-shot* é mais expressivo em modelos de maior escala.

Estratégias mais elaboradas combinam essas modalidades com instruções estruturais. O *Chain-of-Thought* (CoT), proposto por Wei *et al.* (2022), instrui o modelo a externalizar seu raciocínio em passos intermediários antes de produzir a resposta final, o que demonstrou ganhos significativos em tarefas que exigem composição lógica. Em vez de gerar diretamente a saída, o modelo é induzido a "pensar em voz alta", decompondo o problema e justificando decisões parciais, o que tende a reduzir erros em tarefas estruturadas.

No presente trabalho, esses paradigmas constituem objeto direto de investigação experimental. O Estágio 2 da metodologia emprega geração *zero-shot* como linha de base; o Estágio 3 aplica engenharia de *prompt* estruturada, com instruções refinadas, seções nomeadas e restrições de formato; e o Estágio 4 incorpora recuperação aumentada de exemplos via RAG, configurando uma forma de *few-shot* contextualizada, combinada a uma etapa explícita de CoT, em que o agente é instruído a raciocinar sobre a fonte de *log*, os campos relevantes, os modificadores apropriados e a técnica ATT&CK correspondente antes de emitir a regra *Sigma* final. A comparação controlada entre essas estratégias permite quantificar o ganho marginal de cada camada de complexidade, conforme detalhado no Capítulo 4.

2.5 Geração Aumentada por Recuperação (RAG)

Os LLMs apresentam duas limitações estruturais quando aplicados a domínios técnicos especializados. A primeira é o corte de conhecimento (*knowledge cutoff*), já que o modelo só conhece informações disponíveis até a data em que foi treinado, e tudo o que surge depois (um novo CVE, uma nova técnica de ataque, uma atualização de norma) fica fora do seu alcance. A segunda é a tendência à alucinação: na ausência de informação confiável, o modelo pode produzir respostas plausíveis em forma, mas incorretas em conteúdo. Para mitigar ambos os problemas, Lewis *et al.* (2020) introduziram o paradigma da *Retrieval-Augmented Generation*, em português, Geração Aumentada por Recuperação (RAG), que combina o LLM com uma base externa de documentos consultada em tempo de inferência. O funcionamento do RAG pode ser dividido em duas etapas. Primeiro, dado um pedido feito pelo usuário, um mecanismo de recuperação seleciona, de uma base previamente indexada, os documentos mais relevantes para responder àquele pedido. Em seguida, esses documentos recuperados são fornecidos como contexto adicional ao modelo de linguagem, que executa a etapa de geração se apoiando simultaneamente em seu conhecimento interno e nas evidências externas recém-fornecidas (Lewis *et al.*, 2020). Este arranjo implica que o conhecimento do sistema pode ser atualizado sem retreinar o modelo, bastando acrescentar documentos novos à base; que as respostas se tornam rastreáveis aos documentos que as embasaram, oferecendo um caminho para a diminuição de alucinações; e que o domínio de especialização pode ser adaptado com a substituição da base de documentos, embora a qualidade do sistema permaneça fortemente dependente da curadoria e indexação dessa base (Lewis *et al.*, 2020).

A próxima subseção detalha os mecanismos técnicos que tornam essa recuperação possível. Em particular, como textos são convertidos em representações numéricas que permitem comparação semântica. E a subseção seguinte discute aplicações específicas de RAG em cibersegurança.

2.5.1 *Embeddings* e Bancos de Dados Vetoriais

A recuperação eficaz de documentos relevantes depende de quão confiável é a medição do quanto dois textos tratam do mesmo assunto. Buscas baseadas em correspondência literal de palavras-chave costumam falhar, porque dois textos podem descrever o mesmo conceito usando vocabulários diferentes — um analista pode escrever "escalada de privilégios", enquanto a documentação técnica usa "*privilege escalation*". Modelos clássicos de recuperação já tratavam essa limitação representando documentos como vetores em um espaço de termos (*Vector Space Model*), com peso de relevância calculado por esquemas como Frequência de Termo–Frequência Inversa de Documento, do inglês *Term Frequency–Inverse Document Frequency* (TF-IDF) (Manning; Raghavan; Schütze, 2008). Sistemas modernos de RAG, no entanto, recorrem a representações vetoriais densas e contextualizadas (os *embeddings*).

A ideia central dos *embeddings* modernos é que textos com sentido próximo são posicionados próximos uns dos outros no espaço vetorial, mesmo quando usam palavras diferentes. Para gerar essas representações, modelos especializados são treinados a partir de redes neurais *Transformer* adaptadas: Reimers e Gurevych (2019) propuseram o *Sentence-BERT* (SBERT), uma das arquiteturas mais usadas para esee fim, que estende o modelo BERT em uma rede siamesa para produzir, com baixo custo computacional, *embeddings* fixos por sentença. A partir desse trabalho derivou-se uma família de modelos amplamente adotada em sistemas de RAG, incluindo o `all-MiniLM-L6-v2` e o `BAAI/bge-small-en-v1.5` utilizados neste trabalho.

Uma vez que cada documento esteja representado por um vetor, a similaridade entre dois textos pode ser quantificada por uma métrica geométrica. A mais comum é a similaridade de cosseno, que mede o ângulo entre dois vetores: vetores próximos no espaço produzem cosseno próximo de 1 (alta similaridade), enquanto vetores ortogonais resultam em cosseno próximo de 0 (sem relação). Essa métrica é especialmente adequada para a comparação semântica de textos porque ignora a magnitude dos vetores e foca apenas na direção que eles apontam, uma propriedade almejada quando se compara conteúdo semântico em documentos de diferentes comprimentos (Manning; Raghavan; Schütze, 2008). Porém, cabe registrar que a eficácia da recuperação por similaridade depende fortemente da qualidade dos *embeddings* utilizados. Representações inadequadas ao domínio podem aproximar documentos semanticamente distantes, ou afastar documentos relevantes, comprometendo a etapa subsequente de geração.

À medida que a base cresce, percorrer todos os vetores a cada consulta se torna inviável. Bancos de dados vetoriais especializados foram desenvolvidos para resolver esse problema, indexando os *embeddings* em estruturas que permitem a recuperação aproximada dos vizinhos mais próximos em tempo sublinear. Em particular, Johnson, Douze e Jégou (2019) demonstraram que técnicas de busca por similaridade podem ser escaladas para bilhões de vetores quando combinadas a aceleração por *Graphics Processing Unit* (GPU), viabilizando aplicações práticas de RAG em larga escala. Entre as implementações mais utilizadas estão o FAISS, do *Meta AI Research*, e o *ChromaDB*, banco vetorial leve e de código aberto adotado neste trabalho. O *ChromaDB* armazena os *embeddings* das regras *Sigma* indexadas e responde, para cada consulta, com os documentos mais semanticamente próximos, prontos para serem fornecidos como contexto ao *LLM gerador*.

2.5.2 RAG aplicado à Cibersegurança

O paradigma de RAG vem sendo investigado em cibersegurança como uma alternativa promissora à atualização contínua de LLMs por retreinamento. Há razões estruturais do próprio domínio que motivam essa investigação. O cenário de ameaças evolui de forma contínua, com novas vulnerabilidades e técnicas adversárias surgindo diariamente, ritmo que torna eco-

nomicamente inviável manter um LLM atualizado por retreinamento frequente. Além disso, o conhecimento especializado em segurança da informação está distribuído em múltiplas fontes (bases padronizadas, relatórios técnicos, regras públicas de detecção, entre outras), formato adequado a um sistema de recuperação. Reconhecendo essas afinidades, o uso de LLMs em cibersegurança vem crescendo rapidamente, com aplicações documentadas em detecção de ameaças, análise de vulnerabilidades e suporte a SOCs (Hasanov *et al.*, 2024).

Investigações recentes exploram essa direção em diferentes frentes. Rajapaksha, Rani e Karafili (2024) propuseram um sistema de pergunta-resposta baseado em RAG para investigação e atribuição de ataques cibernéticos, reportando redução de alucinações e rastreabilidade das fontes utilizadas na geração das respostas, com desempenho superior (em avaliação sobre sua base de conhecimento curada) ao uso isolado de GPT-3.5 e GPT-4o. Os autores também mostram que o modelo RAG pergunta-resposta gera respostas melhores quando recebe exemplos *few-shot* em vez de instruções *zero-shot*.

Outras aplicações descritas na literatura recente incluem o enriquecimento de inteligência de ameaças, a geração assistida de cenários de simulação de ataque e o suporte automatizado a analistas em SOCs. Cabe notar, contudo, que esses ganhos não são universais: a qualidade da recuperação permanece como gargalo recorrente, podendo deixar de fora o documento correto mesmo quando ele existe na base, e a sensibilidade do método à curadoria do *corpus*, à escolha do modelo de *embeddings* e à formulação da consulta torna o desempenho do RAG fortemente dependente das decisões de projeto.

Neste trabalho, o RAG opera sobre uma base de 3.134 regras *Sigma* do repositório oficial *SigmaHQ*, previamente convertidas em *embeddings* e armazenadas no *ChromaDB*. A cada novo pedido, seja a partir de um CVE, de um *hash*, de um texto sobre uma ameaça recente ou de uma combinação dessas entradas, o agente consulta o banco vetorial e recupera as regras mais semanticamente próximas, que servem como exemplos contextuais durante a geração da nova regra. Esse mecanismo materializa, na prática, a aplicação simultânea das três tarefas clássicas de PLN discutidas na Seção 2.4: recuperação de informação (busca por regras semelhantes), extração de informação (identificação de CVE e hash na entrada) e sumarização (condensação do material coletado em uma descrição factual). A avaliação experimental do impacto efetivo dessa estratégia, comparada a abordagens alternativas, é objeto do Capítulo 5. A integração de todos esses componentes em um único fluxo automatizado é tema da próxima seção, sobre Agentes Autônomos.

2.6 Agentes Autônomos

Um agente autônomo é, na definição clássica de Russell e Norvig (2020), um sistema computacional capaz de perceber seu ambiente por meio de sensores, raciocinar sobre essas percepções e atuar sobre o ambiente por meio de atuadores para alcançar objetivos definidos, com mínima ou nenhuma intervenção humana entre a entrada e a saída. Atualmente, agen-

tes autônomos baseados em LLMs tem se consolidado como uma abordagem promissora, na qual o LLM atua como “núcleo cognitivo” do sistema, sendo responsável pelo raciocínio em linguagem natural, enquanto componentes externos, como ferramentas, memórias e bases de conhecimento, ampliam suas capacidades para além do que o modelo conseguiria realizar isoladamente. Para este tipo de agente, Wang *et al.* (2024) propõem um arcabouço unificado que articula quatro módulos: perfil (a identidade e o papel atribuídos ao agente), memória (representação do contexto e do histórico), planejamento (decomposição de tarefas em passos) e ação (execução por meio de ferramentas).

A distinção entre utilizar um LLM e construir um agente é fundamental para compreender a contribuição do presente trabalho. Em uma chamada direta ao LLM, o usuário envia uma pergunta e obtém uma resposta única e estática, sem que o modelo consulte fontes externas, corrija sua própria saída ou integre múltiplas etapas de raciocínio. Em um agente, por outro lado, o LLM é orquestrado em um fluxo que pode envolver sucessivas chamadas, recuperação de informações externas, validação de resultados intermediários e ajustes iterativos — comportamentos sustentados pelos módulos de memória, planejamento e ação descritos por Wang *et al.* (2024).

As duas subseções seguintes abordam, respectivamente, o padrão de raciocínio *ReAct* e a biblioteca adotada para implementar a arquitetura do agente proposto neste trabalho.

2.6.1 Padrão *ReAct*

O padrão *ReAct* (*Reasoning and Acting*), proposto por Yao *et al.* (2023), é um esquema para a construção de agentes baseados em LLMs. Sua ideia central é intercalar, em uma mesma sequência, traços de raciocínio (*Thought*) e ações (*Action*) sobre ferramentas externas, intercaladas com observações (*Observation*) dos resultados dessas ações. O ciclo opera em três passos, iniciando com o raciocínio do modelo sobre o estado atual do problema, em seguida decide qual ação executar (se consulta uma *Application Programming Interface*, em português Interface de Programação de Aplicações (API) ou faz busca em uma base, por exemplo), e por fim incorpora a observação do resultado ao seu raciocínio subsequente, repetindo o ciclo até concluir a tarefa (Yao *et al.*, 2023).

A contribuição do *ReAct* foi mostrar que combinar raciocínio e ação no mesmo fluxo pode reduzir alucinações e melhorar a precisão em tarefas que exigem informação externa, em comparação ao uso isolado de cada um dos dois mecanismos (Yao *et al.*, 2023). O agente desenvolvido neste trabalho incorpora parcialmente esse paradigma, pois o LLM é instruído a produzir raciocínio explícito antes de gerar a regra *Sigma*, no formato *Thought-Action-Observation* descrito por Yao *et al.* (2023). Porém, diferentemente do *ReAct* original, a sequência de ações do agente como consultar APIs externas, recuperar contexto com RAG, gerar a regra, validar e refazer se necessário, não é decidida dinamicamente pelo modelo, mas orquestrada por uma máquina de estados externa, descrita na próxima subseção.

2.6.2 *LangGraph* como orquestrador

O *LangGraph* é uma biblioteca de código aberto do *framework* de orquestração de código aberto *LangChain* [4], voltada para a construção de agentes em forma de grafos dirigidos (LangChain Team, 2024b). Cada nó do grafo representa uma etapa funcional (uma chamada ao LLM, uma consulta a uma API ou uma rotina de validação) e as arestas determinam a ordem e as condições de transição entre nós. Todos os nós operam sobre um estado compartilhado, tipicamente estruturado como um dicionário tipado (*Graphstate*), que cada etapa pode ler e atualizar. Essa formulação permite representar fluxos com ramificações condicionais, ciclos de correção e roteamento dinâmico, enquanto se mantém controle determinístico sobre a estrutura geral do agente (LangChain, 2024).

O agente proposto neste trabalho é construído sobre esta biblioteca e sua arquitetura se organiza em seis nós sequenciais: classificador de entrada, consulta a APIs externas, recuperação por RAG, geração da regra, validação sintática e julgamento semântico. Todos os nós estão conectados por arestas fixas, exceto entre a validação e a geração, em que uma aresta condicional implementa um ciclo de auto-correção: caso a regra gerada seja rejeitada pelo validador, o estado é devolvido ao nó de geração com o erro registrado, permitindo nova tentativa enriquecida pela informação do erro anterior. Essa configuração caracteriza o sistema como um agente autônomo baseado em estado (*stateful autonomous agent*), conforme a taxonomia de Wang *et al.* (2024), porque possui núcleo de raciocínio (o LLM), memória estruturada (estado compartilhado), acesso a ferramentas externas (APIs e banco vetorial) e capacidade de auto-correção sem intervenção humana. Cabe registrar, no entanto, os limites de autonomia da implementação adotada: a sequência de ferramentas consultadas é fixa, e o agente não implementa experiências entre execuções, características presentes em arquiteturas mais avançadas (agentes treinados) e identificadas como direções de trabalho futuro.

2.7 Métricas de similaridade

Métricas de similaridade são funções matemáticas que recebem dois objetos e produzem um valor numérico que quantifica o quanto eles se parecem. São ferramentas fundamentais em recuperação da informação, mineração de texto e aprendizado de máquina, e cobrem aplicações que vão desde a busca por documentos relevantes em uma base até a comparação entre a saída de um modelo gerador e uma referência de avaliação (Manning; Raghavan; Schütze, 2008). Esta seção apresenta as métricas índice de *Jaccard* e similaridade do cosseno, que serão empregadas neste trabalho para comparar as regras *Sigma* geradas pelo agente proposto contra o conjunto de regras de referência. Elas operam sobre representações distintas dos dados, conjuntos discretos e vetores contínuos, e oferecem perspectivas complementares sobre o conceito de “semelhança”.

[4] www.langchain.com

2.7.1 Índice de *Jaccard*

O índice de *Jaccard* foi proposto pelo botânico suíço Paul *Jaccard* em 1912, no estudo da distribuição de espécies vegetais nos Alpes, como um coeficiente para quantificar a semelhança entre comunidades vegetais a partir das espécies que tinham em comum (Jaccard, 1912). Desde então, foi adotado em diversas áreas da ciência da computação para comparar conjuntos discretos. Dados dois conjuntos finitos A e B, o índice é definido como a razão entre o tamanho da interseção e o tamanho da união dos dois conjuntos:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{1}$$

O valor de $J(A, B)$ varia entre 0 e 1. Quando os conjuntos não compartilham nenhum elemento, a interseção é vazia e o índice vale 0; quando os conjuntos são idênticos, a interseção é igual à união e o índice vale 1; valores intermediários indicam graus parciais de sobreposição.

Para efeitos de ilustração do conceito, considere dois conjuntos de palavras-chave associados a duas regras de detecção: $A = \{\texttt{powershell}, \texttt{encoded}, \texttt{base64}\}$ e $B = \{\texttt{powershell}, \texttt{encoded}, \texttt{obfuscated}, \texttt{macro}\}$. A interseção contém dois elementos (`powershell` e `encoded`), e a união contém cinco; portanto, $J(A, B) = 2/5 = 0{,}4$. A leitura é que as duas regras compartilham 40% dos termos considerados (Manning; Raghavan; Schütze, 2008). Essa interpretabilidade direta torna o *Jaccard* particularmente útil quando se deseja responder a perguntas pontuais sobre quais elementos estão ou não em comum.

2.7.2 Similaridade do Cosseno

A similaridade do cosseno é a métrica padrão em modelos de espaço vetorial, paradigma que representa objetos, tipicamente documentos, como vetores em um espaço de muitas dimensões (Manning; Raghavan; Schütze, 2008).

Dado dois vetores $\mathbf{v}$ e $\mathbf{w}$, a similaridade é definida como o cosseno do ângulo entre eles, calculado pelo produto escalar dos vetores normalizado pelos seus comprimentos (normas):

$$\cos(\mathbf{v}, \mathbf{w}) = \frac{\mathbf{v} \cdot \mathbf{w}}{\|\mathbf{v}\| \, \|\mathbf{w}\|} \tag{2}$$

Para vetores com componentes não negativos, como vetores TF-IDF ou *embeddings* de texto normalizados, o resultado varia entre 0 e 1: vetores que apontam na mesma direção produzem cosseno próximo de 1, o que indica alta similaridade, e vetores ortogonais produzem cosseno próximo de 0, indicando nenhuma relação. A propriedade fundamental dessa métrica é que ela depende apenas da direção dos vetores, ignorando suas magnitudes, aspecto útil na comparação de textos com extensões distintas, nos quais a norma do vetor se altera conforme o comprimento do documento, enquanto a direção reflete o conteúdo semântico de forma con-

sistente (Manning; Raghavan; Schütze, 2008). Por essa razão, a similaridade do cosseno se tornou a métrica de referência em sistemas de recuperação semântica, incluindo o mecanismo de RAG descrito na Seção 2.5.1.

2.7.3 Aplicação

A aplicação das métricas neste trabalho será na etapa de avaliação das regras *Sigma* geradas pelo agente de modo a compará-las com as regras de referência extraídas do repositório oficial *SigmaHQ*. A similaridade do cosseno é calculada entre os *embeddings* da regra gerada e da regra de referência, oferecendo uma medida global de proximidade semântica entre as duas.

Em contrapartida, o índice de *Jaccard*, será aplicado separadamente nos três conjuntos extraídos das regras: as *tags* de técnicas MITRE ATT&CK declaradas, os campos do bloco `detection` e o par `(category, product)` do `logsource`. Apresentar os três valores de *Jaccard* separadamente, em vez de combiná-los em uma única média, revela o *tipo* de erro que a regra gerada apresenta: uma regra pode acertar a fonte de *log* mas declarar técnicas equivocadas, ou fazer o contrário. Apesar de um eventual desempenho médio similar, as duas falham por razões diferentes e exigiriam correções diferentes na prática.

Demais procedimentos operacionais e critérios de interpretação estão detalhados no Capítulo 4.

3 TRABALHOS RELACIONADOS

Este capítulo apresenta uma revisão de estudos correlatos que abordam a automatização da geração de regras de detecção por meio de LLMs e a aplicação de aprendizado de máquina a sistemas SIEM. São discutidos os *frameworks LLMCloudHunter*, *RulePilot* e GRIDAI, cada um representando uma escolha distinta de linguagem de regra e de arquitetura, seguidos de dois estudos que aplicam técnicas de aprendizado de máquina à triagem e análise de *logs* em SIEMs. A leitura conjunta desses trabalhos permite situar o presente estudo em relação ao estado da arte da automação em detecção de ameaças.

Os trabalhos foram selecionados com a ajuda do modelo *Gemini*, sob requisição de estudos que se aproximassem deste.

3.1 *LLMCloudHunter*

O *LLMCloudHunter*, proposto por Schwartz *et al.* (2025), é um *framework* baseado em *GPT-4o* para a geração automatizada de regras *Sigma* a partir de relatórios abertos de inteligência de ameaças (*Open-Source Cyber Threat Intelligence* (OSCTI)) voltados a ambientes em nuvem. A proposta responde a três lacunas identificadas pelos autores em trabalhos anteriores: a ausência de saídas acionáveis prontas para uso em sistemas de segurança, o descarte de informações visuais presentes nos relatórios e o foco em ambientes *on-premises*. A metodologia se articula em três fases. Na primeira, de pré-processamento, o *BeautifulSoup* extrai o texto útil do relatório e um componente de análise de imagens transcreve capturas de tela e diagramas por meio das capacidades multimodais do modelo. Na segunda fase, de processamento em nível de parágrafo, um extrator de chamadas de API opera em dois passos (identificação explícita e inferência de operações de alto nível para chamadas concretas) combinados a uma votação por maioria para mitigar alucinações; em seguida, um extrator de táticas e técnicas mapeia cada chamada ao *MITRE ATT&CK*, um gerador sintetiza as regras *Sigma* candidatas e um validador ajusta a sintaxe ao padrão. A terceira fase, de processamento em nível de OSCTI, consolida os candidatos: um otimizador unifica ou separa campos de seleção conforme a coerência lógica, um refinador elimina redundâncias entre parágrafos e um extrator de IOC incorpora endereços IP e *user agents* identificados no relatório à lógica de detecção.

Avaliado sobre uma base de vinte relatórios reais de ameaças em nuvem, com anotação manual de referência, o *framework* atingiu 83% de precisão e 99% de revocação na identificação de chamadas de API atribuídas ao agente da ameaça, e 99% e 97%, respectivamente, para os indicadores de comprometimento. Do total de regras *Sigma* geradas, 99,18% foram compiladas com sucesso e convertidas em consultas para a plataforma *Splunk*, evidenciando aderência ao padrão sintático. O estudo confirmou ainda a relevância do processamento visual: a interpretação conjunta de texto e imagem mostrou-se estratégia adequada para lidar com a heterogeneidade dos relatórios de ameaças em nuvem (Schwartz *et al.*, 2025, seção 4.3).

3.2 *RulePilot*

O *RulePilot*, proposto por Wang *et al.* (2026), é um agente baseado em LLM para a geração e a conversão automatizada de regras de detecção em linguagens específicas de SIEMs proprietários, com foco no SPL do *Splunk* e suporte adicional à *Kusto Query Language* (KQL) do *Microsoft Sentinel*. Os autores justificam a escolha por linguagens proprietárias sob a argumentação de que formatos abertos como o *Sigma*, embora portáveis, restringem-se majoritariamente à correspondência de padrões por expressões regulares e não suportam agregações estatísticas, junções multi-fonte ou cálculos em janelas temporais; limitações que, segundo os autores, comprometem a detecção de ataques sofisticados envolvendo sequências de eventos, e que atingiriam diretamente trabalhos como o *LLMCloudHunter*. A arquitetura do *framework* se organiza em três componentes. Um módulo de raciocínio em cadeia (CoT) decompõe a descrição de entrada em seis etapas estruturadas, da interpretação dos objetivos de segurança até a otimização da regra. Uma representação intermediária (IR) abstrai a lógica semântica da sintaxe específica do SIEM-alvo, permitindo que o LLM se concentre no raciocínio antes de se comprometer com a sintaxe final. Um mecanismo de reflexão e otimização iterativa avalia a regra em três dimensões — consistência lógica, correção sintática e viabilidade de execução, e dispara refinamentos, incluindo a validação em uma instância real do *Splunk* com *logs* verdadeiros. Para a conversão entre linguagens, o *framework* incorpora um mecanismo de RAG alimentado pela documentação oficial de migração da *Microsoft*, com busca vetorial combinada a re-ranqueamento lexical.

A avaliação foi conduzida com três LLMs como motor (GPT-4o, DeepSeek-V3 e LLaMA-3) sobre 1.699 regras oficiais do repositório *Splunk Security Content* e 229.968 *logs* coletados de 61 testes atômicos do *MITRE ATT&CK*. Os resultados indicam que o *RulePilot* supera os LLMs autônomos em até 107,4% na similaridade textual com as regras de referência, medida por BLEU, ROUGE e METEOR [1], e alcança precisão de 100% em diversas táticas do *MITRE ATT&CK* sob execução real no *Splunk*. Um estudo de ablação confirmou que a representação intermediária e o raciocínio em cadeia com reflexão atuam de forma complementar: a primeira sustenta a correção sintática, o segundo sustenta a consistência lógica e a remoção conjunta produz o pior desempenho. Na tarefa de conversão SPL-KQL, o *framework* atingiu F1 igual a 1,000 em regras simples e 0,919 em regras com junções multi-fonte, sugerindo a viabilidade técnica da migração entre plataformas. Esses ganhos, entretanto, acompanham um custo computacional relevante: cerca de dez vezes mais *tokens* de *prompt* e seis vezes mais tempo de geração que o LLM autônomo, em razão do raciocínio em múltiplas etapas e dos ciclos de refinamento iterativo.

[1] BLEU (Papineni *et al.*, 2002), ROUGE (Lin, 2004) e METEOR (Banerjee; Lavie, 2005) são métricas automáticas que medem a similaridade entre um texto gerado e uma referência pela sobreposição de *n-gramas*. BLEU prioriza precisão e ROUGE prioriza revocação; METEOR acrescenta sinônimos, radicais e ordem das palavras. Valores mais altos indicam maior alinhamento textual.

3.3 GRIDAI

O GRIDAI, proposto por Li *et al.* (2025) (ainda disponível apenas como pré-*print*), é um *framework* de ponta a ponta para a geração e o reparo automatizado de regras de detecção de intrusão em sistemas baseados em assinaturas, como *Suricata* e *Snort*. A lacuna apontada pelos autores em trabalhos anteriores, como AutoCombo (Du; Hu; Hewlett, 2021) e Syrius (Alcantara *et al.*, 2021), é que estes geram uma regra nova para cada amostra de ataque recebida e dependem de tráfego normal como referência, sem verificar se uma regra existente poderia ser simplesmente atualizada — o que produz, com o tempo, acúmulo de redundâncias e alertas duplicados. A contribuição central do GRIDAI é uma decisão prévia à ação: para cada amostra, o sistema avalia se o ataque é inédito, caso em que cria uma nova regra, ou se é variação de algo conhecido, caso em que atualiza a regra existente para cobrir também o novo caso. A arquitetura é organizada em quatro módulos coordenados por uma lógica central. Um agente de avaliação que testa a amostra diretamente no motor de detecção e, apenas em caso de silêncio, consulta o LLM para deliberar; um gerador de novas regras que produz e testa múltiplos candidatos por análise em etapas do conteúdo do ataque; um reparador que garante que a regra modificada permaneça funcional tanto para o ataque original quanto para a nova variante; e um módulo de memória que registra as alterações. E cada regra produzida é validada em tempo real no motor de detecção, com *rollback* da operação caso as falhas ultrapassem um limite pré-definido. Os experimentos foram conduzidos com o GPT-4.1 como motor principal, sobre um conjunto sintético (sete tipos de ataques) e um real (quarenta e três tipos), com replicações em quatro outros LLMs para verificar robustez ao modelo. O sistema atingiu 100% de acerto na decisão sobre o conjunto sintético e 96,9% nos casos de reparo do conjunto real; a acurácia para geração de novas regras foi de 60,5%, resultado que os autores atribuem à similaridade entre regras do conjunto de referência. Comparado a três *baselines*, o GRIDAI detectou 100% dos ataques no conjunto sintético e 95,4% no real, sem falsos alarmes. O estudo de ablação mostrou que a remoção do mecanismo de reparo eleva a taxa de alertas duplicados de 12,7% para 66,8%, evidenciando empiricamente a vantagem de atualizar regras existentes em vez de criar novas a cada variante detectada.

3.4 Aprendizado de máquina aplicado a SIEMs e análise de *logs*

A aplicação de aprendizado de máquina para a análise automatizada de *logs* e a identificação de ameaças foi explorada de forma convergente por Shah *et al.* (2022) e Nurusheva *et al.* (2024), que partem da constatação de que o volume e a heterogeneidade dos *logs* gerados por sistemas modernos inviabilizam a inspeção manual, e de que abordagens baseadas somente em regras estáticas falham na identificação de ataques novos ou ofuscados [2]. Para superar

[2] Ofuscação é a técnica de mascarar a aparência de um código, comando ou tráfego de rede para torná-lo incompreensível aos sistemas de detecção e analistas, sem alterar a função executável original.

essa limitação, os dois estudos recorrem a técnicas de ML, embora com metodologias distintas. Shah *et al.* (2022) propõe um *pipeline* híbrido para o cenário recorrente de ausência de dados rotulados: após pré-processamento textual dos registros e vetorização por TF-IDF, aplica o algoritmo não-supervisionado *K-means* para gerar rótulos iniciais, que subsidiam o treinamento de um classificador supervisionado *Naive Bayes*. Avaliado sobre o *dataset BlueGene/L* (BGL), o modelo alcançou acurácia média de 98% na predição de eventos anômalos, demonstrando viabilidade para ambientes que exigem triagem rápida.

Já Nurusheva *et al.* (2024) propõem a integração direta de algoritmos supervisionados a um sistema SIEM construído sobre a pilha ELK, validada com ciberataques simulados na aplicação propositalmente vulnerável *Damn Vulnerable Web Application* (DVWA) e amostras baseadas no *framework* MITRE. Foram treinados modelos clássicos (*Support Vector Machine*, *Random Forest*, *Decision Tree* e *AdaBoost*) e uma Rede Neural Convolucional, do inglês *Convolutional Neural Network* (CNN) adaptada à classificação de eventos de segurança; os autores relatam ganhos operacionais na correlação de eventos e na automação da resposta, ainda que o estudo tenha caráter predominantemente descritivo, sem métricas comparativas rigorosas. A leitura conjunta das duas pesquisas evidencia um dilema prático na modernização da segurança cibernética —- o equilíbrio entre refinamento algorítmico isolado e aplicabilidade sistêmica. Shah *et al.* (2022) demonstra a viabilidade técnica do componente preditivo em ambiente controlado, enquanto Nurusheva *et al.* (2024) ilustra a integração operacional desses componentes em SIEMs reais, mas raramente ambos os aspectos são tratados de forma articulada em um único estudo.

3.5 Posicionamento do Trabalho Proposto

Os trabalhos analisados nas seções anteriores expõem duas vertentes complementares de pesquisa em automação da detecção de ameaças: uma centrada em técnicas de aprendizado de máquina aplicadas à triagem e classificação de *logs*, sem produzir artefatos sintáticos diretamente acionáveis em SIEMs, na qual se situam Shah *et al.* (2022) e Nurusheva *et al.* (2024); e outra centrada na geração automatizada das próprias regras de detecção, valendo-se das capacidades generativas dos LLMs, na qual se situam o GRIDAI (Li *et al.*, 2025), o *RulePilot* (Wang *et al.*, 2026) e o *LLMCloudHunter* (Schwartz *et al.*, 2025). O presente trabalho se insere na segunda vertente, compartilhando com Schwartz *et al.* (2025) o objetivo de geração de regras *Sigma*, ainda que com diferenças relevantes de escopo — enquanto o *LLMCloudHunter* opera sobre relatórios OSCTI e produz regras específicas para ambientes de nuvem AWS, o presente estudo opera a partir de registros CVE estruturados e notícias de vulnerabilidade, visando regras não restritas a um provedor específico. Já os demais trabalhos, embora voltados a outras linguagens de regra ou não geradores de regras, oferecem contribuições complementares. Li *et al.* (2025) e Wang *et al.* (2026) apresentam precedentes arquiteturais relevantes (respectivamente, a validação iterativa em motor de detecção real e o raciocínio em cadeia com

reflexão), padrões metodológicos próximos da arquitetura *LangGraph* aqui empregada; e Shah *et al.* (2022) e Nurusheva *et al.* (2024) fundamentam a premissa de que técnicas de aprendizado de máquina, já consolidadas na análise de eventos de segurança, estendem-se para tarefas mais sofisticadas como a geração automatizada de regras de detecção.

Vale registrar uma divergência teórica entre os trabalhos analisados: Wang *et al.* (2026) argumentam que o formato *Sigma* é insuficiente para a detecção de ataques sofisticados envolvendo sequências de eventos, por restringir-se sobretudo à correspondência de padrões por expressões regulares, e criticam diretamente trabalhos como o *LLMCloudHunter*. O presente trabalho optou por manter o foco em *Sigma* em decorrência de sua portabilidade entre múltiplos SIEMs, da existência do repositório público *SigmaHQ* (base do mecanismo de RAG aqui empregado) e de sua posição como padrão aberto mantido pela comunidade de *detection engineering*, com clara consciência das limitações expressivas apontadas por Wang *et al.* (2026) e das próprias limitações que a arquitetura RAG sobre o *SigmaHQ* pode apresentar na tarefa específica de geração de regras *Sigma*, questão que integra o objeto empírico deste estudo. Nesse contexto, permanece em aberto a questão metodológica de quanto a complexidade arquitetural, desde a invocação direta de uma LLM em formato *zero-shot* até a orquestração de um agente autônomo com RAG e validação iterativa, efetivamente contribui para a qualidade das regras geradas a partir de fontes públicas de inteligência sobre ameaças. Diferentemente dos cinco trabalhos analisados, que tipicamente comparam uma única abordagem proposta contra LLMs autônomos ou métodos baseados em mineração de dados, o presente estudo investiga essa questão por meio da comparação controlada entre as três estratégias incrementais *zero-shot*, engenharia de *prompt* e agente autônomo com RAG, buscando quantificar empiricamente o ganho marginal proporcionado por cada camada de complexidade arquitetural.

4 MATERIAIS E MÉTODOS

Este capítulo descreve os materiais e os métodos que serão utilizados para conduzir esta pesquisa. Apresentar-se-ão o conjunto de ferramentas e bibliotecas empregadas no desenvolvimento, a base de dados de regras *Sigma*, as três estratégias de geração de regras que constituem o objeto de comparação deste trabalho (geração *zero-shot*, engenharia de *prompt* e agente autônomo com RAG), as métricas de avaliação adotadas e, por fim, a formulação das hipóteses e o procedimento de análise estatística.

4.1 Materiais

Quadro 1 – Modelos de linguagem utilizados no trabalho

Ferramenta	Versão	Disponível em	Finalidade
ChatGPT	—	chat.openai.com/	LLM comercial utilizada nas fases de geração *zero-shot* e engenharia de *prompt*.
Gemini	—	gemini.google.com/	Idem; versão Pro usada como auxílio para geração de texto para este trabalho.
Claude	—	claude.ai/	Idem.
Microsoft Copilot	—	copilot.microsoft.com/	Idem.
DeepSeek	—	chat.deepseek.com/	Idem.
Llama 3.1 Instruct	8B	ollama.com/library/ llama3.1	LLM geradora de regras *Sigma*.
Qwen 2.5	1.5B	ollama.com/library/ qwen2.5	LLM usada no destilador de texto.
Qwen 2.5	14B	ollama.com/library/ qwen2.5:14b	LLM usada como juiz das regras.
bge-small-en-v1.5	—	huggingface.co/BAAI/ bge-small-en-v1.5	Modelo de *embeddings*.
bge-reranker-base	—	huggingface.co/BAAI/ bge-reranker-base	Modelo *cross-encoder*.

Fonte: Autoria própria (2026).

Conforme apresentado no Quadro 4, para o desenvolvimento do trabalho se utilizou dois equipamentos. O *laptop*, com GPU integrada e recursos de memória mais limitados, foi utilizado no início do trabalho em virtude de problemas técnicos com o *desktop*. Graças a este incidente, a fase de prototipação do agente se deu com o modelo *Qwen 2.5 1.5B*, que demanda menos computacionalmente e permitiu iterar rapidamente sobre a arquitetura. O *desktop*, equipado com GPU dedicada, foi empregado na execução dos experimentos finais e na geração das regras *Sigma* com o modelo *Llama 3.1 8B*.

Os materiais apresentados no Quadro 1, atendem a duas finalidades distintas. As LLMs comerciais *ChatGPT*, *Gemini*, *Claude*, *Microsoft Copilot* e *DeepSeek* foram utilizadas exclusiva-

Quadro 2 – *Frameworks*, banco vetorial e ambiente de desenvolvimento

Ferramenta	Versão	Disponível em	Finalidade
langchain-core	1.4.1	pypi.org/project/langchain-core/	Fornece as abstrações de mensagens utilizadas pelo gerador.
langchain-community	0.4.2	pypi.org/project/langchain-community/	Fornece os *loaders* de documentos utilizados no banco vetorial.
langchain-ollama	1.1.0	pypi.org/project/langchain-ollama/	Fornece a classe `ChatOllama`.
langchain-chroma	1.1.0	pypi.org/project/langchain-chroma/	Utilizada pelo mecanismo de RAG.
langchain-huggingface	1.2.2	pypi.org/project/langchain-huggingface/	Carrega o modelo de *embeddings*.
langgraph	1.2.4	langchain-ai.github.io/langgraph/	Biblioteca para modelar o agente como uma máquina de estados.
sentence-transformers	5.5.1	sbert.net/	Biblioteca para carregar os modelos de *embeddings* e o *cross-encoder*.
Ollama	—	ollama.com/	Servidor local para inferência de LLMs de código aberto.
chromadb	1.5.9	trychroma.com/	Banco de dados vetorial.
Visual Studio Code		code.visualstudio.com/	Ambiente de desenvolvimento.

Fonte: Autoria própria (2026).

Quadro 3 – Bibliotecas de validação, coleta de dados e análise estatística

Ferramenta	Versão	Disponível em	Finalidade
Python	3.12.3	python.org/	Linguagem utilizada para implementação.
pySigma	1.3.3	github.com/SigmaHQ/pySigma	Biblioteca oficial utilizada na validação semântica das regras geradas.
PyYAML	6.0.3	pyyaml.org/	Biblioteca para (*parsing*) de YAML.
requests	2.34.2	requests.readthedocs.io/	Requisições HTTP a fontes públicas de inteligência sobre ameaças.
beautifulsoup4	4.15.0	crummy.com/software/BeautifulSoup/	Biblioteca para *parsing* de HTML.
ddgs	9.14.4	pypi.org/project/ddgs/	Busca *web* complementar.

Fonte: Autoria própria (2026).

mente nas duas primeiras fases do experimento (geração *zero-shot* e engenharia de *prompt*), em que cada modelo recebe o mesmo conjunto de entradas e gera regras *Sigma* de forma independente, permitindo a comparação entre diferentes provedores. Já os modelos de código aberto (*Llama 3.1* e *Qwen 2.5*), juntamente com os *frameworks LangChain*, *LangGraph* e *ChromaDB*, constituem a infraestrutura do agente autônomo desenvolvido na terceira fase, sendo responsáveis pela geração, recuperação de contexto via RAG, e validação iterativa das regras. Os modelos de *embeddings* e *re-ranking* são componentes internos do agente, utilizados na etapa de recuperação e não participam diretamente da geração de texto.

Quadro 4 – *Hardware* utilizado no desenvolvimento do trabalho

Componente	Desktop (principal)	Laptop (prototipação)
Modelo	ASUS TUF H310M-PLUS GAMING/BR	Lenovo IdeaPad 3 15ALC6
Processador	Intel Core i7-9700K @ 3.60 GHz (8 núcleos)	AMD Ryzen 7 5700U @ 1.80 GHz (8 núcleos / 16 *threads*)
Memória RAM	16 GB DDR4 2133 MHz (2×8 GB)	12 GB DDR4 3200 MHz (8 GB + 4 GB)
GPU	NVIDIA GeForce RTX 3060 (12 GB GDDR6 dedicados)	AMD Radeon Lucienne (integrada, sem VRAM dedicada)
Armazenamento	SSD Kingston SA400S3 480 GB (SATA)	SSD NVMe SSSTC 512 GB
Sistema operacional	Linux Ubuntu 24.04 LTS	Linux Ubuntu 24.04 LTS

Fonte: Autoria própria (2026).

4.1.1 Base de Dados

Quadro 5 – Composição das bases de dados utilizadas no trabalho

Conjunto	Quantidade	% do *SigmaHQ*	Origem
Subconjuntos derivados do repositório *SigmaHQ* válidas (3.748 regras)			
Base de teste (`test_cases`)	50	1,34%	Selecionada por amostragem estratificada *round-robin* a partir do repositório *SigmaHQ*.
Base de teste estendida (`extra_test_cases`)	10	0,27%	Idem.
Base de conhecimento RAG (`rag_knowledge`)	3.134	83,62%	Idem.
Subtotal selecionado	**3.194**	**85,2%**	
Regras não utilizadas	554	14,8%	Restante do repositório, não incluído nos subconjuntos.
Regras geradas pelas LLMs comerciais (a partir de `test_cases`)			
Subconjunto	**Quantidade total**	**Quantidade pós seleção**	
Regras geradas com prompt *zero-shot*	250	50	Regras de `test_cases` replicadas por 5 LLMs comerciais.
Regras geradas com engenharia de *prompt*	250	50	Idem.
Regras geradas pelo agente autônomo	50	50	Regras de `test_cases` processadas pelo agente *LangGraph*.
Subtotal de regras geradas	**550**	**150**	

Fonte: Autoria própria (2026).

A fundação da base de dados do trabalho é o repositório oficial *SigmaHQ*. No entanto, apenas os cinco subdiretórios interessam: `rules`, `rules-compliance`, `rules-emerging-threats`, `rules-placeholder` e `rules-threat-hunting`,

pois são eles que contém as regras-alvo. Esta distinção é feita no código, que poderá ser conferido no repositório do *Github* `agente-autonomo-rag-regras-sigma` [1].

Para elaborar as avaliações a partir das gerações das regras, selecionou-se uma base de teste oficial (*test_cases*) composta por 50 regras oficiais e um conjunto de teste extra com outras 10 regras. Este último selecionado para testes repetitivos de aferição durante o desenvolvimento do agente. Decidiu-se por retirar, ao todo, 85% de regras do repositório *Sigma* [2] para uso no trabalho. Sendo assim, para o desenvolvimento do agente com RAG, que requer uma base de conhecimento (*rag_knowledge*), foram destinados 83,62% das regras selecionadas (Tabela 5).

Como já citado anteriormente, para confrontar a qualidade das regras geradas pelo agente desenvolvido, empregou-se a avaliação de cada regra do conjunto *test_cases* por cinco LLMs comerciais. Este arranjo de regras é dividido em quatro grupos, sendo o primeiro composto pelas 50 regras originais do repositório oficial *SigmaHQ*, que servem de referência para todas as comparações subsequentes. O segundo grupo é composto pelas 250 regras-produto da replicação pelas cinco LLMs comerciais, utilizando *prompts zero-shot*. No grupo três, o procedimento foi repetido nas mesmas cinco LLMs, porém com *prompts* elaborados com *prompt engineering*, resultando em mais 250 regras. Por fim, o quarto grupo é composto pelas 50 regras geradas pelo agente autônomo.

Para as avaliações foi realizada uma filtragem das melhores regras dos grupos 2 e 3, utilizando a técnica de *LLM-as-a-judge*, almejando 150 regras no total, como pode ser visto na Tabela 5. Este último processo é explicado com mais detalhes na Seção 4.2.2.4.

4.2 Métodos

A seguir serão apresentadas as etapas que compõem o método utilizado nesta pesquisa, detalhando cada fase do processo, desde a seleção e divisão do acervo de regras *Sigma*, o desenvolvimento do agente proposto, a composição comparativa entre as estratégias de geração investigadas e o protocolo de avaliação que será aplicado às regras geradas.

4.2.1 Seleção dos dados

A separação dos dados em conjuntos de treino e teste é um requisito metodológico fundamental para garantir a validade científica da comparação entre os quatro estágios desta pesquisa, em particular do Estágio 4. Neste trabalho, como não há treinamento de fato, as bases dividem-se entre base de recuperação e teste. Em sistemas baseados em RAG, o risco de *data leakage*, definido como “a introdução não intencional de informação preditiva sobre o alvo proveniente do processo de coleta, agregação ou preparação dos dados” por Kaufman *et*

[1] Disponível em: www.github.comdaninotagente-autonomo-rag-regras-sigma.git

[2] Disponível em www.github.com/SigmaHQ/sigma

al. (2012), pode se manifestar de forma particularmente traiçoeira: caso a mesma regra figure simultaneamente na base recuperável pelo agente e no gabarito contra o qual sua saída será comparada (regras originais), o agente poderia, durante a etapa de *retrieval*, recuperar a própria regra de referência como contexto, invalidando as métricas de similaridade e produzindo uma superestimação artificial em seu desempenho. Esse problema é amplamente reconhecido na literatura como uma das causas mais frequentes de resultados inflados em *pipelines* de ML (Kaufman *et al.*, 2012).

Para evitar este cenário, as regras utilizadas como gabarito de teste (conjuntos `test_cases` e `extra_test_cases`) foram fisicamente removidas do conjunto de regras disponibilizado ao RAG do agente antes da seleção deste último (`rag_knowledge`), de modo que nenhuma regra do teste pode ser recuperada como exemplo durante a geração, na execução do agente. A divisão foi implementada por meio do *script* `gerador_datasets_unificado.py` [3], desenvolvido especificamente para este trabalho.

A varredura se restringe aos cinco subdiretórios do repositório oficial *SigmaHQ* que concentram as regras de produção (seção 4.1.1), excluindo-se de forma deliberada o diretório *deprecated* (já que contém regras descontinuadas), bem como pastas de exemplos, documentação e ferramentas auxiliares contidas no repositório.

A seleção das regras foi feita por amostragem estratificada balanceada com base no *logsource*, e não por amostragem aleatória simples. A partir de cada regra do repositório, o algoritmo deriva uma assinatura que consiste na tripla `categoria_produto_serviço`, oriunda do campo `logsource` (por exemplo, `process_creation_windows` ou `network_connection_linux`), e então agrupa as regras conforme estas categorias. Em seguida, a seleção acontece via *round-robin*, onde a ordem das categorias é inicialmente embaralhada, para eliminar algum viés de ordenação do dicionário; o algoritmo, então, percorre ciclicamente as categorias embaralhadas e, a cada passo, sorteia uma regra aleatória pertencente à categoria vigente, até se atingir a quantidade desejada (50 para o `test_cases`, por exemplo). Esse procedimento garante que os conjuntos resultantes contemplem uma maior diversidade possível de fontes de *log* dentro do conjunto amostral, com o objetivo de evitar a sobre-representação de categorias numerosas (como *Windows process creation*) que ocorreria sob amostragem aleatória simples — a aparição de mais regras *Windows* é justificável sob a lógica de que existem mais regras deste tipo no repositório *Sigma* original.

O processo produz três conjuntos, descritos na Tabela 5. O conjunto `test_cases` é o *baseline* de referência (gabarito, padrão-ouro) para os quatro estágios; o conjunto *extra_test_cases*, foi usado para testes e tomadas de decisões durante a construção do agente; e o conjunto *rag_knowledge*, também é selecionado via *round-robin* e constitui a base de conhecimento consultada pelo agente do Estágio 4 (Seção 4.2.2.5) durante a etapa de *retrieval*. As regras remanescentes não são utilizadas em nenhum dos estágios, ficando reservadas como margem para experimentos posteriores.

[3] Disponível em: www.github.com/daninot/agente-autonomo-rag-regras-sigma/tree/main/src.

4.2.2 Geração das regras para a comparação entre estágios

A avaliação proposta neste trabalho compara estratégias distintas de obtenção de regras *Sigma* e as separa em quatro estágios. O primeiro estágio é a coleta direta de regras escritas por especialistas humanos, as regras de referência, direto do repositório oficial *SigmaHQ*; o segundo consiste na geração automática de regras por LLMs comerciais sob instrução mínima (*zero-shot*); o terceiro consiste na geração de regras pelas mesmas LLMs usando *prompts* elaborados com *prompt engineering* e no quarto a geração de regras é feita pelo agente autônomo composto de RAG e ferramentas externas, desenvolvido neste trabalho. Cada estágio é melhor descrito a seguir conforme a estrutura: objetivo, *input* ou entrada, processo e saída, de modo a facilitar a comparação direta entre eles. As regras produzidas pelos Estágios 2 e 3 foram submetidas a uma filtragem por *LLM-as-a-judge*, conforme detalhado na Seção 4.2.2.4, restando ao final 150 regras geradas para comparação com as 50 regras de referência.

4.2.2.1 Estágio 1: regras originais do repositório *SigmaHQ*

O Estágio 1 não constitui propriamente uma estratégia de geração, mas a referência contra a qual as demais serão comparadas. Seu objetivo é fornecer um conjunto de 50 regras *Sigma* convencionadas como corretas por construção — escritas, revisadas e mantidas pela comunidade especializada do projeto *SigmaHQ*, que servirão como gabarito para todas as métricas de avaliação descritas na Seção 4.2.4. Como entrada, este estágio utiliza diretamente o repositório oficial *SigmaHQ*, conforme descrito na Seção 4.1.1. Aqui, o processo consiste exclusivamente na seleção das 50 regras que compõem o conjunto *test_cases*, conduzida pelo *script* `gerador_datasets_unificado.py` segundo o procedimento de amostragem apresentado na Seção 4.2.1. A saída deste estágio são as 50 regras armazenadas no diretório `data/test_cases/`, cada qual em um arquivo `.yml` nomeado segundo a convenção do repositório de origem. Essas regras servem tanto de gabarito para o cálculo das métricas de similaridade e correspondência estrutural quanto para fornecer as referências externas (campos `references`, `description` e `tags`) que são extraídas e utilizadas como base para os *prompts* dos Estágios 2, 3 e 4.

4.2.2.2 Estágio 2: geração *zero-shot* com LLMs comerciais

O Estágio 2 tem como objetivo reproduzir regras *Sigma* a partir de LLMs comerciais sob instrução mínima, replicando o cenário no qual um analista de segurança pode recorrer a uma LLM de uso geral sem elaborar *prompts* complexos. O paradigma adotado é o de *zero-shot prompting*, definido por Brown *et al.* (2020) como a ocasião em que “o modelo recebe apenas uma descrição em linguagem natural da tarefa, sem qualquer demonstração ou exemplo”. Trata-se da forma mais simples e direta de interação com uma LLM: o *prompt* contém apenas a

instrução do que deve ser feito, cabendo ao modelo inferir, a partir de seu treinamento prévio, como executar a tarefa.

A entrada, para cada um dos 50 cenários (regras-alvo do conjunto *test_cases*),é um *prompt zero-shot* (`prompt1.txt`) curto e padronizado, contendo apenas o texto "*Generate ONE SIGMA rule (https://sigmahq.io/docs/basics/rules.html) for this event:*" seguido do(s) *link*(s) presentes no campo `references` da regra original. Não são fornecidos exemplos de regras *Sigma* previamente escritas, instruções sobre a estrutura YAML esperada, nem orientações específicas sobre o uso de *tags MITRE ATT&CK* ou definição de `logsource`. No entanto, pode ser que as LLMs façam busca na internet, o que as confere vantagem no enriquecimento das informações.

Então são submetidos cada um dos 50 `prompt1` para as cinco LLMs comerciais diferentes, totalizando 250 interações independentes. Cada LLM é instruída a gerar uma única regra *Sigma* válida em formato `YAML`. A diversidade de modelos comerciais foi adotada para evitar que os resultados reflitam particularidades de uma única arquitetura ou política de treinamento. Portanto, a saída deste estágio são 250 regras *Sigma*, arquivos `YAML` dentro do diretório de cada caso de teste, junto à respectiva regra original e ao *prompt* utilizado. Essas regras compõem, juntamente com as do Estágio 3, o conjunto candidato à filtragem por *LLM-as-a-judge* descrita na Seção 4.2.2.4.

4.2.2.3 Estágio 3: geração com *prompt engineering*

O propósito do Estágio 3 é investigar em que medida a aplicação sistemática de *prompt engineering*, conforme conceituado na Seção 2.4.3, eleva a qualidade das regras geradas pelas mesmas LLMs comerciais do Estágio 2. Ao contrário do paradigma *zero-shot* adotado no Estágio 2, em que a instrução é mínima e o modelo opera apenas com seu conhecimento prévio, no Estágio 3 a instrução é enriquecida com elementos que orientam o modelo quanto ao papel a ser assumido, ao formato esperado da saída e ao contexto técnico da ameaça.

Neste estágio a entrada é, para cada um dos 50 cenários, um *prompt* construído a partir de um *template* padronizado no qual variam apenas as informações específicas da ameaça. Este *template* foi elaborado com o auxílio do modelo *Claude* em processo iterativo, enviando a regra original *Sigma* e solicitando o retorno do *prompt2* correspondente, que incorpora cinco técnicas reconhecidas na literatura (Liu *et al.*, 2023):

- Atribuição de papel: o *prompt* inicia instruindo o modelo a operar como se fosse um "*Senior Threat Detection Engineer*", engenheiro sênior de detecção de ameaças, em inglês;
- Restrição de formato de saída: a instrução determina que o modelo produza exatamente uma regra *Sigma* válida em YAML;
- Ancoragem por referência: inclui-se a URL da especificação oficial *Sigma*;

- Injeção de contexto estruturado: o conteúdo da requisição é organizado em seções nomeadas — `Threat to detect`, `Target environment` e `References`;
- Enriquecimento de domínio: além das referências externas, o *prompt* fornece descrição textual da ameaça, identificação do produto/serviço alvo e categorização de fonte de *log* esperada.

Assim como no estágio anterior, cada um dos 50 *prompts* elaborados é submetido manualmente às cinco LLMs comerciais escolhidas, totalizando 250 interações independentes. Os Estágios 2 e 3 compartilham deliberadamente os mesmos modelos, regras-alvo e canal de submissão; a única variável que muda é a estratégia de *prompting*, de modo que as diferenças observadas nos resultados podem ser atribuídas exclusivamente a esse fator. Por fim, a saída deste estágio são 250 regras *Sigma*, armazenadas no mesmo diretório de cada caso de teste paralelo às regras do Estágio 2. Em conjunto com as regras do capítulo anterior, essas regras compõem o universo de 500 candidatos que serão submetidos a um processo de filtragem por *LLM-as-a-judge*, explicado na subseção seguinte.

4.2.2.4 Filtragem

Após a geração dos 500 cenários com as cinco LLMs comerciais, 250 para o *prompt1* e 250 para o *prompt2*, foi necessário filtrar as 50 melhores de cada conjunto, para viabilizar a comparação com os cenários disponíveis. Os cinco LLMs foram os próprios juízes e para isso foi necessário organizar um teste minimamente cego, alterando o nome das regras para letras de A a E. Para cada regra original, em cada uma das cinco LLM, o *input* foi: as 5 regras (`A.yml`, `B.yml`, `C.yml`, `D.yml` e `E.yml`), a regra original e um *prompt* de instrução:

prompt filtro

```
You are a Senior Threat Detection Engineer.
Five attached files (A, B, C, D, E) are Sigma rule candidates
generated for the same threat.
The reference rule from SigmaHQ and the original prompt are also
attached. Pick the BEST candidate based on:
1. Syntactic validity (well-formed YAML, required fields, follows
Sigma pec);
2. Structural correctness (correct logsource, valid MITRE tags,
appropriate detection approach);
3. Detection quality (covers the threat, avoids
over/underdetection). Output ONLY:
WINNER: <letter>
WHY: <one sentence>
```

Após o modelo devolver a resposta de qual considera melhor, segundo os critérios do *prompt*, os resultados foram anotados manualmente em uma tabela. Depois de contabilizar qual LLM recebeu mais votos por regra, obteve-se uma regra vencedora, para todas as regras do Estágio 2 e 3.

Todos os *prompts* e tabelas com o processo da filtragem por *LLM-as-a-judge* estão disponíveis no repositório `agente-autonomo-rag-regras-sigma` [4].

4.2.2.5 Estágio 4: geração via agente com RAG

O quarto e último estágio tem como finalidade gerar regras *Sigma* por meio do agente autônomo desenvolvido neste trabalho. Diferentemente dos Estágios 2 e 3, que recorrem a LLMs comerciais de grande porte acessadas via interface *web*, no Estágio 4 há o emprego de um modelo de linguagem aberto e de menor porte, executado localmente, cuja saída é apoiada por recuperação de contexto (RAG), consulta a fontes externas de inteligência de ameaças e validação iterativa. A descrição detalhada da arquitetura do agente, de seus componentes e funcionamento interno é apresentada na Seção 4.2.3; nesta seção, descreve-se apenas o papel do agente como uma das estratégias de geração avaliadas.

Neste estágio, se utilizam três entradas de *prompts* para cada um dos 50 cenários: *prompt1*, *prompt2* e *prompt3*. As duas primeiras correspondem, respectivamente, ao *prompt zero-shot* do Estágio 2 e ao *prompt* com *prompt engineering* do Estágio 3; a terceira, *prompt3*, é uma versão refinada do *prompt2* em texto corrido, sem divisão em seções nomeadas. A supressão dos cabeçalhos estruturais (`Threat to detect`, `Target environment` e

[4] www.github.com/daninot/agente-autonomo-rag-regras-sigma/blob/main/data

`References`) já é implantada no código: o agente realiza internamente, por meio de seus nós de classificação e enriquecimento, a função de identificar o tipo de entrada, consultar fontes externas e recuperar exemplos relevantes, tornando redundante a estruturação manual exigida das LLMs comerciais nos estágios anteriores. O *prompt3* é considerado a configuração principal do agente, mas as variantes *prompt1* e *prompt2* são executadas em paralelo para subsidiar a análise complementar de sensibilidade entre os três.

O processo de geração é totalmente automatizado e conduzido pelo *script* `sigma_agent_automatico.py` [5], que submete cada uma das 50 regras-alvo às três variantes de *prompt*, totalizando 150 execuções independentes do agente. Cada execução produz sete arquivos: uma regra gerada para cada *prompt* e o parecer do juiz semântico do nó 6 sobre essa regra, em formato JSON, e um arquivo `.csv` centralizado de metadados, com informações sobre a configuração da execução e seus resultados quantitativos. A estrutura completa desse `.csv` e o uso de cada um de seus campos para as análises subsequentes são descritos no Capítulo 5.

A saída deste estágio são 150 regras *Sigma*, das quais 50 foram geradas a partir do *prompt3* e compõem o conjunto oficial do Estágio 4 a ser confrontado com o gabarito de referência. As 100 regras restantes, provindas de *prompt1* e *prompt2*, não participam da comparação entre os quatro estágios, mas são preservadas para a análise de sensibilidade entre os três tipos de *prompts*.

A partir do Estágio 4 se desenrolam dois eixos de análise: a comparação entre as regras dos quatro estágios e a análise de sensibilidade do agente ao *prompt*, que examina, mantidos constantes o modelo e a arquitetura, em que medida a variação da formulação da entrada (do *prompt1* ao *prompt3*) afeta a qualidade e a taxa de aprovação das regras geradas. O detalhamento das métricas aplicadas em cada eixo é apresentado na Seção 4.2.4.

4.2.3 Desenvolvimento do agente

4.2.3.1 Visão geral da arquitetura

O agente desenvolvido neste trabalho é implementado conforme o paradigma de *graph-based agent*, no qual o fluxo de execução é representado por um grafo dirigido cujos nós correspondem a etapas funcionais distintas e as arestas correspondentes determinam a ordem e as condições de transição no fluxo. A implementação utiliza a biblioteca *LangGraph* (LangChain Team, 2024a), parte do ecossistema *LangChain*, que oferece componentes fundamentais para a definição de máquinas de estados executadas sobre um estado compartilhado mutável (*GraphState*). Cada nó recebe o estado, executa sua tarefa, atualiza os campos pertinentes e o devolve ao grafo, que decide o próximo nó a ser executado com base em arestas fixas ou condicionais.

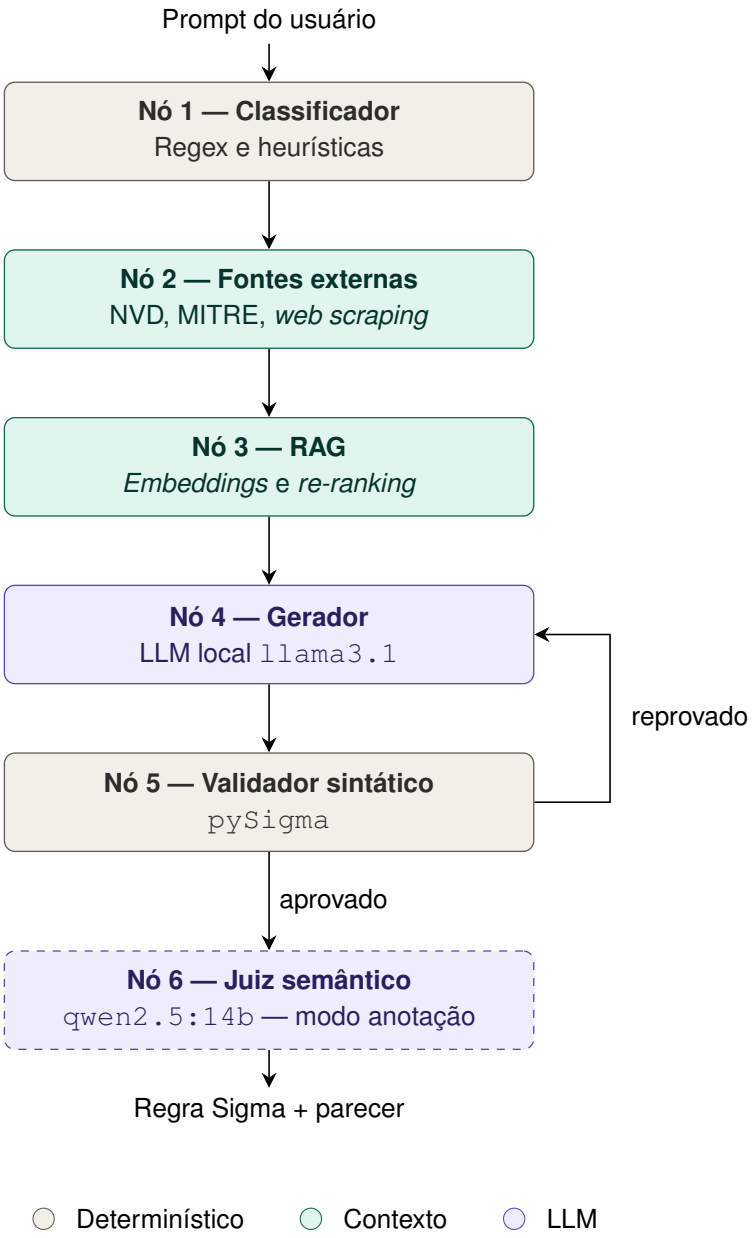
[5] Disponível em www.github.com/daninot/agente-autonomo-rag-regras-sigma

A arquitetura se articula em torno de seis nós, organizados em um *pipeline* de enriquecimento progressivo da informação. O nó 1, é determinístico (baseado em expressões regulares e heurísticas, sem invocação de LLM) e interpreta o *prompt* recebido, identifica seu tipo, extrai *Uniform Resource Locators* (URL)s de referência e isola a descrição textual da ameaça. O nó 2, enriquece o estado com dados técnicos obtidos junto a APIs de inteligência de ameaças (NVD, MITRE ATT&CK) e, quando necessário, recorre a *web scraping* complementado por uma subrotina de destilação por LLM. O nó 3 consulta um banco vetorial contendo exemplos de regras *Sigma* previamente embarcadas, aplicando recuperação semântica por similaridade seguida de *re-ranking* por *cross-encoder*. O nó 4 invoca uma LLM local (*Llama* 3.1) que sintetiza a regra *Sigma* em YAML a partir do estado acumulado. O nó 5 submete a regra gerada à ferramenta *pySigma* e emite veredito de aprovação ou reprovação. E por fim, o nó 6, é um juiz semântico que opera em modo anotação, ou seja, registra um parecer qualitativo sobre a regra, sem capacidade de bloqueá-la ou desencadear nova geração.

A ordem dos nós não é arbitrária. A consulta às APIs externas no nó 2 precede deliberadamente a recuperação por RAG (nó 3) para que o material técnico retornado pelas APIs (por exemplo, a descrição oficial de um CVE no NVD) sirva de consulta semântica enriquecida ao banco vetorial, aumentando a relevância dos exemplos recuperados. A geração, que acontece no nó 4, ocorre apenas após o estado conter as três camadas de contexto anteriores e é seguida imediatamente pela validação sintática do nó 5, de modo que regras malformadas sejam corrigidas. O grafo incorpora uma aresta condicional após o nó 5: caso a regra reprove na validação sintática, o controle retorna ao nó 4 para nova tentativa de geração, configurando um laço de auto-correção limitado por um número máximo de tentativas; Trata-se da única ramificação não-linear do grafo.

A Figura 3 apresenta a topologia completa da máquina de estados, com a identificação dos seis nós, suas conexões e a aresta condicional. O nó 6, com borda tracejada, registra parecer do juiz semântico sem bloquear o fluxo.

Figura 3 – Arquitetura do agente



Fonte: Autoria própria (2026).

As arestas definem o caminho percorrido pelo estado durante a execução. Como ilustrado na Figura 3, o agente utiliza dois tipos de conexão: conexões fixas, que sempre levam ao próximo nó previamente definido, e uma única conexão condicional, onde o destino depende do conteúdo do estado naquele momento (LangChain, 2024). Cinco conexões fixas ligam o ponto de partida do nó 1 até o nó 5. Após o nó 5, encontra-se a conexão condicional, cuja decisão é tomada por uma função auxiliar chamada `roteador_de_validacao`, descrita a seguir. Por fim, uma última conexão fixa leva o nó 6 ao ponto final do grafo: o produto do agente, a regra *Sigma* (Figura 3).

A função `roteador_de_validacao` segue três regras, verificadas em ordem. Primeira: se o campo `erro_validacao` contém o valor `APROVADO`, o controle segue para o nó

6 e o laço termina com sucesso. Segunda: se o erro persiste, mas o contador de `tentativas` já contabiliza 4, o controle também segue para o nó 6, encerrando o laço por esgotamento das tentativas. Entretanto, a regra considerada sintaticamente inválida ainda é gravada em disco (o *script* não a descarta nem ignora) e avaliada pelo juiz semântico, cujo parecer também é salvo para análise posterior. Terceiro: em qualquer outra situação, quando o *pySigma* é executado sobre a regra e detecta um erro, o controle volta ao nó 4 para uma nova tentativa de geração, agora orientada pela mensagem de erro guardada em `erro_validacao`. O limite de quatro tentativas é uma escolha de projeto: alto o suficiente para permitir a convergência em casos recuperáveis, mas baixo o suficiente para impedir laços infinitos diante de *prompts* que não podem ser corrigidos.

Vale destacar que essa conexão condicional é a única ramificação do grafo. Todas as outras decisões internas do agente, qual API consultar no nó 2, quantos exemplos recuperar no nó 3 e qual modelo invocar no nó 4, ocorrem dentro dos próprios nós, sem afetar a estrutura do grafo. Essa separação entre a lógica de fluxo (representada pelo grafo) e a lógica interna (escondida dentro de cada nó) é uma decisão deliberada: ela permite que cada nó seja modificado, substituído ou melhorado de forma independente, sem alterar a arquitetura geral do agente.

As subseções seguintes detalham cada nó: a máquina de estados propriamente dita e o estado compartilhado (4.2.3.2), a classificação das entradas (4.2.3.3), as ferramentas externas (4.2.3.4), o RAG (4.2.3.4), o laço de validação sintática (4.2.3.6 e 4.2.3.7) e a avaliação semântica complementar (4.2.3.8).

4.2.3.2 Máquina de estados (*LangGraph*)

A máquina de estados do agente é construída como um grafo dirigido sobre uma estrutura de dados única e compartilhada, chamada *GraphState*. Trata-se de um dicionário tipado em *Python* (*TypedDict*) que funciona como um “caderno de anotações” visível a todos os nós: cada nó lê os campos de que precisa, executa sua tarefa e devolve apenas os campos que atualizou (LangChain, 2024). Essa organização evita passar informações entre nós por meio de parâmetros explícitos, facilita a manutenção do agente e permite inspecionar todo o seu estado em qualquer momento da execução.

O *GraphState* possui 12 campos, que podem ser agrupados em cinco categorias de acordo com a etapa do *pipeline* em que são preenchidos, conforme a Tabela 1.

Tabela 1 – Campos do `GraphState` agrupados por etapa do *pipeline*

Etapa	Campo	Descrição
Entrada bruta	`input_usuario`	*Prompt input*
Classificação (nó 1)	`tipo_input`	CVE, *hash* ou texto livre
	`termo_busca`	CVE ou *hash* extraído
	`url_fornecida`	URLs identificadas no *prompt*
	`texto_para_rag`	Descrição da ameaça pré-processada
	`contexto_pobre`	Indicador de contexto textual insuficiente
Enriquecimento (nós 2 e 3)	`contexto_api`	Dados técnicos de NVD/MITRE
	`contexto_rag`	Exemplos recuperados do banco vetorial
Geração e validação (nós 4 e 5)	`regra_gerada`	Regra *Sigma* em YAML
	`erro_validacao`	Mensagem do *pySigma* ou `APROVADO`
	`tentativas`	Contador do laço de correção
Avaliação semântica (nó 6)	`parecer_juiz`	Parecer do juiz em bloco JSON com cabeçalho identificador

Fonte: Autoria própria (2026).

4.2.3.3 Nó 1: classificação da entrada

O nó 1 existe para transformar a entrada bruta do usuário em uma estrutura padronizada antes que qualquer etapa baseada em modelo de linguagem seja acionada. Trata-se de um classificador determinístico, isto é, que opera por regras fixas de manipulação de texto (expressões regulares e funções auxiliares), sem recorrer a nenhum LLM. Sua função no fluxo é de recepção e triagem, garantindo que os nós subsequentes recebam campos previsíveis e limpos, independentemente de o *prompt* ter chegado em formato estruturado, com cabeçalhos de seção ou como texto corrido.

O nó detecta primeiro o regime do *prompt* (estruturado ou solto) e, a partir dele, extrai as URLs presentes, separando as que apontam para a ameaça daquelas que remetem apenas à documentação de sintaxe *Sigma*, descartadas por não constituírem inteligência sobre a ameaça. Em seguida, identifica no texto a presença de um identificador CVE ou de um *hash*, classificando a entrada como `cve`, `hash` ou `texto_livre`. Por fim, monta uma descrição da ameaça depurada de URLs e de frases-instrução, destinada a servir como consulta semântica ao mecanismo de recuperação, e avalia se essa descrição é suficiente ou se a informação relevante permanece inacessível, contida apenas nos links fornecidos.

Como saída, o nó 1 entrega ao restante do agente cinco campos que atualizam o estado compartilhado: `tipo_input`, com a categoria da entrada; `termo_busca`, com o CVE ou *hash* isolado, quando presente; `url_fornecida`, com as URLs da ameaça já filtradas; `texto_para_rag`, com a descrição depurada; e `contexto_pobre`, um sinalizador que

indica quando a descrição textual é insuficiente. Esse último campo é determinante para o comportamento do nó 2, que, diante de contexto pobre, aciona uma etapa adicional de coleta e destilação para recuperar o sinal semântico ausente.

4.2.3.4 Nó 2: ferramentas externas

O nó 2 enriquece o estado do agente com informação técnica obtida de fontes externas. Sua existência atende a duas necessidades, e a primeira é de conteúdo: identificadores como CVEs e *hashes* são apenas códigos, sem descrição da ameaça que representam. Para que o gerador (nó 4) escreva uma regra adequada, é preciso traduzi-los em informação técnica — vetor de ataque, plataformas afetadas, técnicas MITRE ATT&CK e nível de gravidade. Já a segunda necessidade, surge quando o *prompt* é pobre em texto, contendo apenas um *link* para um boletim de segurança: nesse caso, o agente precisa reconstruir uma descrição utilizável a partir do material bruto coletado. O nó atende às duas em uma única passagem, organizada nas quatro etapas do Algoritmo 1.

Algorithm 1 Coleta de contexto técnico (nó 2)

Input: Estado $estado$ com campos `url_fornecida`, `tipo_input`, `termo_busca`, `contexto_pobre`
Output: Campo $estado$.`contexto_api` atualizado no estado

trechos ← [ ] *cves*, *cwes* ← extrair_referencias($estado$.`input_usuario`)
`// Etapa 1: URL scraping`
foreach *url* ∈ $estado$.`url_fornecida` **do**
  html ← HttpGet(*url*) **if** *request succeeded* **then**
    texto ← extrair_elementos_semanticos(*html*) adicionar *texto* to *trechos*
  else
    palavras ← extrair_palavras_do_slug(*url*) adicionar *palavras* to *trechos*
  end
  atualizar *cves*, *cwes* com referências encontradas em *url*
end
`// Etapa 2: APIs estruturadas para cada CVE`
foreach *cve* ∈ *cves* **do**
  adicionar consulta_mitre(*cve*) to *trechos* adicionar consulta_nvd(*cve*) to *trechos*
end
if $estado$.`tipo_input` = `hash` **then**
  adicionar consulta_virustotal($estado$.`termo_busca`) to *trechos* `// simulada nesta versão`
end
`// Etapa 3: busca complementar na web`
texto ← $estado$.`termo_busca` primeiro não vazio, else $estado$.`texto_para_rag` adicionar SearchDuckDuckGo(*texto*) to *trechos*
$estado$.`contexto_api` ← concatenar(*trechos*)
`// Etapa 4: destilação condicional por LLM`
if $estado$.`contexto_pobre` ***and*** *tamanho*($estado$.`contexto_api`) ≥ `LIMIAR` **then**
  descricao ← destilar_com_LLM($estado$.`contexto_api`) `// qwen2.5:7b`
  $estado$.`contexto_api` ← *descricao* ‖ $estado$.`contexto_api` `// prepende destilacao`
end
return $estado$.`contexto_api`

A Etapa 1 extrai o conteúdo das URLs fornecidas. Para cada uma, o agente faz uma requisição HTTP e processa o HTML com a biblioteca *BeautifulSoup*, priorizando os elementos mais informativos da página, como parágrafos, cabeçalhos, células de tabela e itens de lista. Essa escolha se justifica porque documentações de auditoria de *logs* costumam expor os nomes reais dos campos em cabeçalhos e tabelas, e são esses nomes que o nó 4 precisará para escrever um bloco `detection` compatível com a fonte de *log*. Se a requisição falhar, um *fallback* extrai palavras-chave do *slug* da URL, preservando pistas sobre a ameaça.

A Etapa 2 consulta bases estruturadas de inteligência de ameaças. Para cada CVE, o agente acessa a API do *MITRE ATT&CK* (vetor de ataque e técnicas associadas) e do NVD

(descrição oficial, CWEs e pontuação CVSS). Quando a entrada é um *hash*, o fluxo previsto seria a consulta ao *VirusTotal* [6] , simulada nesta versão com dados fictícios. A integração real não foi implementada porque a plataforma exige chave de acesso pessoal e impõe limites de uso, e porque entradas do tipo *hash* são raras no conjunto observado, predominando `cve` e `texto_livre`; a opção foi mantida para representar o uso realista por um analista.

A terceira etapa é uma busca complementar na *web* aberta via *DuckDuckGo* [7] , com até cinco resultados e não substitui as fontes estruturadas, pois funciona como rede de segurança para captar blogs técnicos, relatórios de empresas de segurança e boletins de fornecedores, que frequentemente publicam antes das bases oficiais. O termo consultado é o identificador isolado, quando disponível, ou o texto pré-processado pelo nó 1, truncado em 200 caracteres. O *DuckDuckGo* foi escolhido por não exigir chave de API e ser compatível com a biblioteca cliente do projeto.

A última etapa é condicional. Quando o nó 1 sinaliza `contexto_pobre` e o material acumulado atinge o mínimo definido por `LIMIAR_MATERIA_PRIMA`, o agente aciona uma destilação por LLM: o material bruto é enviado ao modelo `qwen2.5:7b`, que produz uma descrição resumida da ameaça, colocada no topo do campo `contexto_api`, acima do material bruto. A escolha do `qwen2.5:7b` (família distinta da do gerador `Llama3.1`) se dá porque em *pipelines* sequenciais, erros das etapas iniciais tendem a se propagar e acumular nas posteriores (Caselli *et al.*, 2015), e usar o mesmo modelo nas duas pontas amplificaria esse risco, por compartilharem os mesmos pontos-cegos de treinamento. Famílias treinadas por equipes e *corpora* diferentes reduzem essa correlação de erros.

Ao final das quatro etapas, o campo `contexto_api` reúne todo o conteúdo técnico coletado, com as descrições estruturadas (MITRE, NVD, destilação) priorizadas sobre o material bruto. Essa ordenação beneficia o nó 3 (RAG), que usa o campo para construir a consulta ao banco vetorial: blocos estruturados geram *embeddings* de qualidade superior à do texto extraído diretamente de páginas HTML.

4.2.3.5 Nó 3: RAG

O nó 3 implementa a etapa de RAG do agente e seu propósito é entregar ao gerador (nó 4) um pequeno conjunto de regras *Sigma* já existentes, semanticamente próximas da ameaça em questão. Essas regras (contidas em `rag_knowledge`) não são fornecidas como "a resposta correta", pois em geral tratam de outras ameaças, mas como referência de formatação e contexto semântico, indicando quais estruturas YAML costumam funcionar bem em casos similares.

A consulta enviada ao banco vetorial é construída segundo uma ordem de prioridade de quatro estratégias, aplicadas conforme o que está disponível no estado. A preferencial combina

[6] Disponível em: www.virustotal.com.
[7] Disponível em: www.duckduckgo.com.

o sinal mais informativo do usuário (o termo isolado, como um CVE, ou as seções técnicas extraídas pelo nó 1) com a descrição técnica recuperada das APIs no nó 2. Na ausência dessa descrição, recorre-se apenas às seções técnicas do nó 1; na ausência também destas, ao termo isolado; e, em último caso, ao *prompt* original do usuário. A ordem reflete uma hierarquia de qualidade: quanto mais rica a consulta, maior a chance de recuperar exemplos alinhados à ameaça real.

A recuperação segue um funil de duas etapas, técnica conhecida como *retrieve-and-rerank* (Nogueira; Cho, 2019). Na primeira, a consulta é convertida em um vetor numérico (*embedding*) pelo modelo `BAAI/bge-small-en-v1.5` (Xiao *et al.*, 2023) e comparada por similaridade de cosseno contra os vetores das regras de `rag_knowledge`, previamente armazenados em um banco vetorial [8] ; dessa busca ampla resultam vinte candidatos. Na segunda, os vinte candidatos passam por um modelo de *re-ranking* do tipo *cross-encoder* (`BAAI/bge-reranker-base`), que avalia cada par consulta-candidato de forma conjunta e produz uma pontuação de relevância refinada; os cinco de maior pontuação são gravados no campo `contexto_rag`.

O funil combina dois paradigmas complementares. O modelo de *embedding* codifica cada texto de forma independente, o que permite pré-calcular e armazenar os vetores de forma rápida, porém com risco de perder nuances da interação entre consulta e documento. Já o *cross-encoder* processa consulta e candidato lado a lado, capturando essas nuances a um custo computacional maior (Nogueira; Cho, 2019). A composição aproveita o melhor de cada um: a busca por *embedding* reduz o banco inteiro a vinte candidatos em milissegundos, e o *re-ranker* aplica seu poder discriminativo apenas sobre esse subconjunto.

Por fim, o nó 3 executa um diagnóstico de coerência sobre os cinco exemplos selecionados, sem interferir no fluxo do agente. O *script* extrai o campo `logsource` de cada exemplo e contabiliza quantas categorias e produtos distintos aparecem entre eles; quando três ou mais categorias coexistem, registra um aviso, pois há risco de a LLM misturar campos de fontes de *log* incompatíveis ao redigir a regra. O diagnóstico não bloqueia a geração, mas serve como sinal de alerta e como registro da qualidade do *retrieval* para diferentes tipos de entrada.

4.2.3.6 Nó 4: geração da regra

O nó 4 produz a regra *Sigma* propriamente dita. Sua entrada é o estado já enriquecido pelos nós anteriores: o nó 1 determinou o tipo da entrada e isolou o termo de busca; o nó 2 acumulou contexto técnico em `contexto_api` e o nó 3 recuperou cinco exemplos de regras reais em `contexto_rag`. Cabe ao nó 4 sintetizar essas informações em um único arquivo YAML conforme a especificação *Sigma*. A tarefa é executada pelo modelo *Llama3.1*, rodado

[8] O banco vetorial utilizado é o *ChromaDB* (www.trychroma.com), uma das implementações *open-source* disponíveis para essa finalidade.

localmente via *Ollama* com temperatura 0,1 — valor baixo, escolhido para favorecer saídas determinísticas e reduzir a variabilidade entre execuções.

A invocação da LLM adota a estrutura dual de *prompt*, com uma mensagem de sistema e uma mensagem de usuário. A mensagem de sistema é carregada de um arquivo externo [9] e reúne as instruções permanentes que regem o comportamento do modelo independentemente da ameaça: estrutura obrigatória do YAML, *tags* válidas, regras de uso dos *logsources*, modificadores aceitos pela especificação, padrões proibidos e diretrizes contra a invenção de campos ou URLs. Esse arquivo foi desenvolvido iterativamente a partir da análise de erros observados no conjunto *extra_test_cases* durante a implementação do agente, e *congelado* antes do início dos experimentos sobre o conjunto *test_cases*, evitando contaminação metodológica entre o ajuste do agente e sua avaliação. A mensagem de usuário, ao contrário, é construída a cada chamada e reúne seis blocos sequenciais:

1. *USER REQUEST*, contendo o *prompt* original do usuário;
2. *USER-PROVIDED REFERENCES*, com a lista exata de URLs autorizadas para o campo `references:` da regra, ou uma instrução explícita para omitir o campo quando nenhuma URL foi fornecida;
3. *FORMATTING TEMPLATE*, com os cinco exemplos recuperados pelo RAG, usados como referência estrutural de formatação;
4. *ADDITIONAL TECHNICAL CONTEXT*, com a informação técnica acumulada pelo nó 2 — MITRE, NVD, busca *web* e, quando aplicável, a descrição destilada da ameaça;
5. *REASONING STEP*, que solicita à LLM a produção de um bloco `thinking...thinking` com raciocínio explícito sobre quatro aspectos críticos da regra a ser gerada — escolha do *logsource*, identificação dos campos de *log*, escolha dos modificadores YAML adequados e mapeamento à técnica MITRE ATT&CK correspondente;
6. *OUTPUT REQUIREMENTS*, com as instruções formais sobre o formato esperado da saída.

O bloco de raciocínio solicitado antes da geração do YAML (item 5) implementa uma forma de *Chain-of-Thought Prompting* (Wei *et al.*, 2022), técnica reconhecida por melhorar saídas que envolvem decisões em múltiplas etapas. Em regras *Sigma*, essas decisões são interdependentes, visto que a escolha do *logsource* restringe os campos disponíveis, que por sua vez determinam quais modificadores fazem sentido. Ao deliberar sobre cada etapa antes de produzir o resultado, a LLM evita se comprometer com uma estrutura inconsistente logo de saída.

[9] Disponível em: https://github.com/daninot/agente-autonomo-rag-regras-sigma/tree/main/prompts

Por fim, o nó 4 incorpora um mecanismo de autocorreção orientada por erro. A partir da segunda tentativa, quando o validador sintático (nó 5) rejeitou a geração anterior e armazenou a mensagem no campo `erro_validacao`, o *prompt* de usuário recebe um bloco adicional, *ATTENTION*, com o erro específico e a instrução para corrigi-lo. Cada nova geração não é, portanto, cega: a LLM recebe *feedback* estruturado e concentra a correção no problema apontado, em vez de gerar a regra novamente do zero. O laço completo entre os nós 4 e 5 é detalhado na próxima subseção.

4.2.3.7 Nó 5: validação sintática iterativa e laço de correção

O nó 5 submete a regra produzida pelo nó 4 a um conjunto de verificações. Seu propósito é aprovar regras que cumprem a especificação oficial *Sigma* ou, em caso de erro, devolver ao gerador uma mensagem técnica precisa que oriente a próxima tentativa. É deste nó que parte o laço de autocorreção entre os nós 4 e 5, descrito ao final desta subseção.

Antes das verificações, o nó 5 executa um pré-processamento sobre a saída bruta da LLM. Quatro normalizações são aplicadas em sequência:

1. Remoção de eventuais cercas de *markdown* (**"'`**) que a LLM possa ter adicionado em torno do YAML;
2. Extração e descarte do bloco `thinking ...thinking`, no qual o nó 4 instruiu a LLM a registrar seu raciocínio passo a passo (Seção 4.2.3.6);
3. Tratamento do caso em que a LLM produz mais de um documento YAML separados por `--`, mantendo-se apenas o primeiro; e
4. Substituição do identificador `id:` da regra por um número único gerado pelo *Python* (*Universally Unique Identifier* (UUID)). Este passo é necessário porque o *system prompt* instrui a LLM que preencha o campo `id:` com um valor temporário qualquer, deixando a geração do identificador real para esta etapa.

Normalizado o YAML, a validação se organiza em três etapas progressivas, da mais barata à mais custosa. A primeira usa a biblioteca *PyYAML* [10] para verificar se o texto é sintaticamente válido. A segunda avalia a estrutura mínima da regra: confirma que o resultado é um dicionário, que os três campos obrigatórios (`title`, `logsource` e `detection`) estão presentes e que toda *tag* declarada pertence à taxonomia oficial do MITRE ATT&CK (Strom *et al.*, 2018). A terceira, mais rigorosa, emprega a biblioteca *pySigma* [11], mantida pelo *SigmaHQ*, que valida a regra contra a especificação completa da plataforma. A regra só é aprovada se passar nas três etapas; em qualquer falha, a mensagem de erro correspondente é gravada no campo `erro_validacao`.

[10] Disponível em: www.pyyaml.org.
[11] Disponível em: www.github.com/SigmaHQ/pySigma.

O fluxo seguinte é decidido por uma função de roteamento que lê o resultado da validação, com três caminhos possíveis: se a regra foi aprovada, o controle segue para o juiz semântico do nó 6, encerrando o laço; se foi reprovada mas o contador `tentativas` atingiu o limite de quatro, o controle também segue para o nó 6, encerrando o laço por exaustão; nas demais reprovações, o controle retorna ao nó 4. Esse retorno não é mera repetição: como descrito na Seção 4.2.3.6, o nó 4 lê o campo `erro_validacao` e anexa a mensagem ao *prompt*, de modo que a LLM receba *feedback* estruturado sobre o problema específico da tentativa anterior.

O ciclo entre os nós 4 e 5 constitui um mecanismo de autocorreção iterativa orientada por *feedback* externo, estudado na literatura sob o nome *self-refinement* (Madaan *et al.*, 2023). O laço aqui difere do mecanismo original no aspecto de *Self-Refine* clássico, em que o próprio modelo gera tanto a saída quanto a crítica que a refina. Aqui, o *feedback* vem de um validador externo, determinístico e auditável (o *pySigma*), o que torna o sinal de correção independente de eventuais vieses da LLM. Esse laço, contudo, protege apenas contra erros sintáticos e estruturais; erros semânticos (uma regra bem formada, mas inadequada à ameaça) exigem a camada adicional de avaliação do nó 6 (Seção 4.2.3.8).

Além disso, também é encaminhado ao nó 6 as regras reprovadas por exaustão, ao invés de descartá-las: isso permite que toda regra gerada seja salva e analisada pelo juiz semântico, viabilizando a caracterização empírica dos modos de falha do agente.

4.2.3.8 Nó 6: avaliação semântica por modelo

O nó 6 implementa uma camada de avaliação complementar à validação sintática do nó 5. O *pySigma* aprova regras *formalmente* corretas, que respeitam a especificação *Sigma* em estrutura, campos, modificadores e operadores, porém não avalia se a regra é adequada à ameaça do cenário, e isso pode levar uma regra sintaticamente impecável a monitorar a fonte de *logs* errada, empregar campos irrelevantes ao ataque descrito ou inventar filtros sem respaldo na descrição original. Captar esse tipo de inadequação, que escapa à validação sintática, é a função do nó 6.

A técnica adotada é o *LLM-as-a-Judge* (Zheng *et al.*, 2023), na qual um LLM dedicado recebe a regra, a descrição original da ameaça e a especificação oficial do *Sigma* e emite um parecer sobre a correção semântica da regra. O modelo empregado é o `Qwen2.5:14b`, executado localmente com temperatura zero, valor que maximiza o determinismo e elimina a variabilidade entre execuções. A escolha do modelo foi feita com base no uso de família arquitetural distinta da do gerador (`Llama3.1`), pelos motivos discutidos na Seção 4.2.3.4, e na maior escala do modelo, adotada porque a avaliação crítica exige mais capacidade de raciocínio do que a geração condicionada.

Como já mencionado, o nó 6 opera em modo anotação, portanto seu parecer não bloqueia o fluxo do agente nem dispara nova rodada de geração, mesmo quando a regra é considerada problemática. O parecer [12] é estruturado em seis dimensões:

1. *logsource*, que avalia se a fonte de *log* declarada usa corretamente os campos *category*, *product* e *service*, e se os valores escolhidos são válidos;
2. *modifiers*, que avalia se os operadores `|contains`, `|endswith` e `|startswith` foram usados de forma adequada;
3. *condition*, que avalia se a lógica *booleana* é sintaticamente válida e faz sentido;
4. *structure*, que avalia a presença dos campos obrigatórios da especificação;
5. *semantic_alignment*, que avalia se a lógica de detecção efetivamente captura a ameaça descrita;
6. *invented_filters*, que avalia se a regra adicionou filtros (caminhos ou *hosts* específicos) que não têm respaldo na descrição original da ameaça.

Cada uma classificada como *PASS*, *WARN* ou *FAIL* e acompanhada de justificativa em uma ou duas frases. A combinação das seis dimensões produz um veredito final com três valores: *APPROVED* (todas as dimensões PASS), *APPROVED_WITH_WARNINGS* (ao menos uma WARN, nenhuma FAIL) e *REJECTED* (ao menos uma FAIL). Por fim, um campo de observações gerais registra um comentário breve sobre a qualidade global da regra e o arquivo do parecer é gravado em disco no formato JSON. Esses arquivos são fonte de evidência para a caracterização dos modos de falha do agente, na Seção de Resultados (5).

4.2.4 Métricas de Avaliação

A avaliação do desempenho do agente e a comparação entre os quatro estágios de geração descritos na Seção 4.2.2 se apoiam em métricas determinísticas, semânticas, de similaridade com o gabarito e testes estatísticos para comparação entre estágios. Todas elas têm como fonte primária de dados o registro de metadados produzido automaticamente pelo ambiente de execução do agente, descrito a seguir.

4.2.4.1 Metadados

Durante a execução do agente com as 50 regras do conjunto *test_cases*, cada combinação (`cenário`, *promptN*) produz 9 objetos salvos no mesmo diretório, conforme descrito

[12] Exemplos disponíveis em: https://github.com/daninot/agente-autonomo-rag-regras-sigma/tree/main/data/test_cases/12%20net_dns_katz_stealer_domain

na Seção 4.2.2.5. Para cada um dos três *prompts* há uma regra *Sigma* gerada, um parecer do juiz semântico e uma linha em um arquivo denominado `metadados_geracao.csv`, que centraliza os dados das 150 execuções. Esta estrutura de dados viabiliza todas as análises subsequentes, visto que cada uma de suas linhas corresponde a uma execução completa do agente, e suas colunas registram os atributos relevantes da execução.

As colunas do registro de metadados podem ser agrupadas conforme suas funções. As colunas `cenario_id` e `prompt_usado` identificam as execuções; as colunas `tipo_input`, `contexto_pobre` e `usou_web_search` registram as decisões tomadas pelos nós 1 e 2 sobre o tratamento da entrada; as colunas `status` (`APROVADO`, `REPROVADO` ou `ERRO_EXECUCAO`), `tentativas` (número de iterações no laço nó 4 – nó 5), `tempo_total_s` e `erro_validacao_final` registram resultados quantitativos da execução; a coluna `veredito_juiz`, seis colunas `juiz_dimensão` , uma para cada dimensão avaliada e `motivo_reprovacao` fazem parte da avaliação semântica do nó 6; e as colunas `regra_referencia`, `arquivo_regra_gerada` e `arquivo_parecer` rastreiam os arquivos gerados. Essa estrutura permite que cada métrica seja calculada a partir de operações simples de filtragem, contagem ou agregação sobre as 150 linhas do `.csv` correspondentes.

4.2.4.2 Métricas determinísticas

O primeiro grupo avalia o desempenho sintático e estrutural do agente, com base nas decisões determinísticas do nó 5. Quatro indicadores são calculados sobre o arquivo `metadados_geracao.csv`:

1. Taxa de aprovação sintática: a proporção de execuções com `status = APROVADO` sobre o total;
2. Número médio de tentativas até a aprovação ou o esgotamento do limite, calculado sobre a coluna `tentativas`;
3. Distribuição dos modos de falha: agrupamento das execuções `REPROVADO` pela mensagem em `erro_validacao_final`, classificadas nas três etapas internas do nó 5 — erro de *parsing* YAML, ausência de campo obrigatório ou erro semântico do *pySigma*; e
4. Tempo médio por execução, calculado sobre a coluna `tempo_total_s`.

4.2.4.3 Métricas semânticas

O segundo grupo de métricas provém do parecer do juiz semântico, do nó 6, e cobre aspectos distintos da conformidade sintática. São calculados três indicadores:

1. Distribuição de vereditos: proporções de regras classificadas como *APPROVED*, *APPROVED_WITH_WARNINGS* e *REJECTED*, obtidas da coluna `veredito_juiz`;

2. Taxa de *PASS* por dimensão: calculada para cada uma das seis colunas `juiz_dimensão`, permite identificar em quais aspectos o agente é mais ou menos sólido. Por exemplo, uma taxa sistematicamente menor em `semantic_alignment` do que em `structure` indica regras bem formadas, porém desalinhadas com a ameaça;

3. Divergência entre nós 5 e 6: proporção de regras aprovadas pelo validador sintático (`status = APROVADO`) que receberam *REJECTED* ou *APPROVED_WITH_WARNINGS* do juiz, quantificando o descolamento entre conformidade formal e adequação semântica.

4.2.4.4 Similaridade com o gabarito e acerto estrutural

Por fim, a medição da proximidade entre cada regra gerada e a regra de referência, em comparação direta um-a-um. Diferentemente dos grupos anteriores, que avaliam atributos absolutos de cada regra, aqui o cálculo é relativo ao gabarito, sob duas perspectivas complementares.

A similaridade semântica geral é medida pela similaridade do cosseno entre os *embeddings* das duas regras, calculados com o mesmo modelo `BAAI/bge-small-en-v1.5` empregado pelo nó 3 (Seção 4.2.3.5). O resultado, entre 1 e 1, aproxima-se de 1 quando as regras são semanticamente próximas. Se duas regras que descrevem ameaças relacionadas terão *embeddings* próximos, ainda que escrevam campos individuais de formas diferentes.

O acerto estrutural é medido pelo índice de *Jaccard* sobre conjuntos derivados de campos-chave, comparados par a par. Enquanto o cosseno opera sobre vetores contínuos e fornece uma visão global, o *Jaccard* opera sobre conjuntos discretos e responde a perguntas pontuais sobre quais elementos coincidem. Três comparações são reportadas separadamente: *Jaccard* sobre as *tags* declaradas (se o agente identificou as técnicas MITRE ATT&CK corretas); sobre os campos do bloco `detection` (se escolheu os campos corretos para a detecção); e sobre o par (*category*, *product*) do `logsource` (se identificou a fonte de *log* adequada). O reporte separado distingue, por exemplo, regras que erram a fonte de *log* mas acertam as *tags* daquelas que fazem o oposto.

As duas métricas são usadas em conjunto por complementaridade: o cosseno mede proximidade mesmo quando as regras não compartilham campos exatos e o *Jaccard* responde se um campo específico foi acertado.

5 RESULTADOS

5.1 Produtos obtidos

Esta seção apresenta a composição final do conjunto experimental empregado nas análises.

A Tabela 2 sintetiza a quantidade de regras *Sigma* em cada um dos quatro grupos comparativos, bem como sua origem e o respectivo procedimento de obtenção.

Tabela 2 – Composição final dos quatro grupos comparativos

Estágio	Designação	N	Origem	Procedimento de obtenção
1	Regras originais	50	Repositório SigmaHQ (test_cases)	Amostragem estratificada *round-robin*
2	Geração *zero-shot*	50	5 LLMs comerciais $\times$ 50 cenários	Filtragem *best-of-N* por LLM-as-a-judge
3	Engenharia de *prompt*	50	5 LLMs comerciais $\times$ 50 cenários	Filtragem *best-of-N* por LLM-as-a-judge
4	Agente autônomo com RAG	50	Agente *LangGraph* (prompt3)	Execução única por cenário
Total		**200**		

Fonte: Autoria própria (2026).

A execução do agente sobre os 50 cenários do conjunto *test_cases*, nas três variações de *prompt*, produziu 150 regras *Sigma* (50 por variação) e os 150 pareceres correspondentes do juiz semântico (nó 6), um para cada regra gerada. Regras e pareceres também foram consolidados no arquivo `metadados_geracao.csv`, que serve de base às análises das seções seguintes.

Para garantir que houve diversificação na seleção estratificada com *round-robin*, verificou-se a representatividade do conjunto de teste sob os aspectos da categoria de fonte de *log*, da tática MITRE ATT&CK declarada nas *tags* e do nível de severidade declarado no campo *level*. A Tabela 3 apresenta as distribuições observadas em cada dimensão.

Tabela 3 – Diversidade da seleção de regras *Sigma* por *round-robin*

Dimensão / Categoria	**N**
(i) Fonte de *log*	
Endpoint Windows	17
Aplicação	13
Rede e *proxy*	9
Cloud / SaaS	7
Endpoint Linux/macOS	4
Total	**50**
(ii) Tática MITRE ATT&CK	
initial-access	15
defense-evasion	14
privilege-escalation	12
execution	11
persistence	9
command-and-control	5
outras (7 categorias)	15
Total	**81**
(iii) Severidade (level)	
medium	30
high	16
critical	2
low	2
Total	**50**

Quanto à família de fonte de *log*, a distribuição provavelmente reflete a composição do repositório *SigmaHQ*, predominantemente voltado a ambientes corporativos, cobrindo as cinco classes mais relevantes de fontes de telemetria em SOCs modernos. As táticas MITRE ATT&CK mais frequentes são *initial-access*, *defense-evasion* e *privilege-escalation*, que figuram entre as três táticas mais observadas em incidentes segundo o último Picus Security (2026). Já no diz respeito à severidade, a maior parcela das regras estão classificadas como *medium* ou *high*.

Por fim, como já mencionado anteriormente, uma execução de `sigma_agent_automatico.py` produz um arquivo `metadados_unificado.csv` [1] que contém um registro com os dados de execução e avaliação semântica.

5.1.1 Desempenho determinístico do agente

Das 150 execuções do agente (50 cenários $\times$ 3 variações de *prompt*), 75 resultaram em regras sintaticamente válidas segundo o nó 5, uma taxa global de aprovação de 50,0%. Nenhuma execução terminou em erro de infraestrutura (`ERRO_EXECUCAO = 0`), o que indica estabilidade da execução local ao longo dos cinco lotes.

[1] Conferir em www.github.com/daninot/agente-autonomo-rag-regras-sigma/tree/main/data/analises

Conforme a Tabela 4, a taxa de aprovação decresce à medida que o *prompt* solicita saídas mais elaboradas: o *prompt1*, mais direto, exibiu a maior estabilidade sintática, e o *prompt3*, mais exigente, a menor. Esse ordenamento levanta a hipótese de que *prompts* que pedem regras mais ricas ampliam tanto os acertos quanto as falhas, deslocando o equilíbrio entre validade formal e sofisticação da detecção.

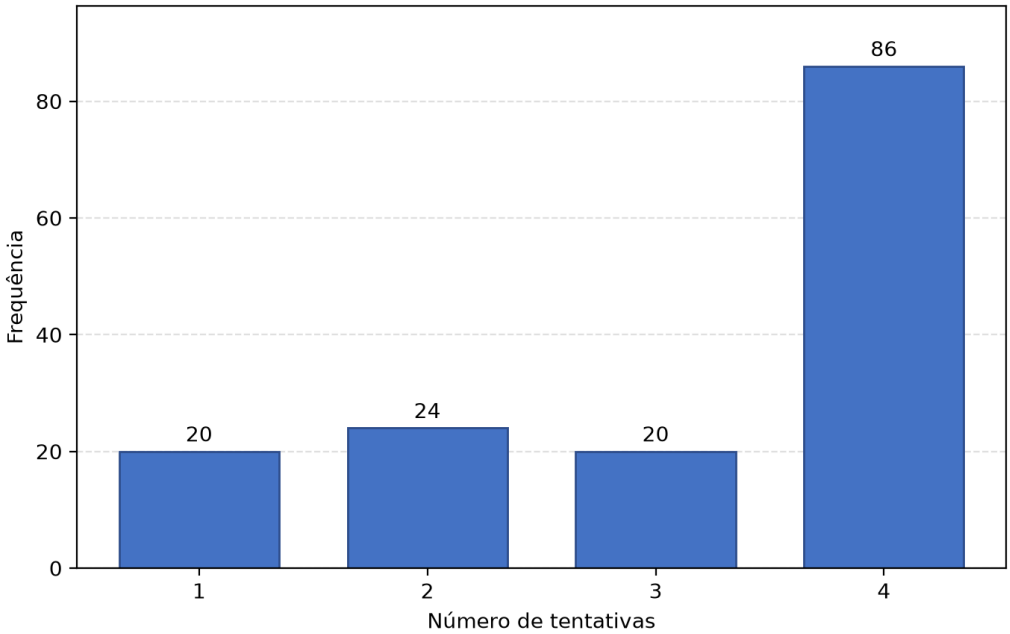
Tabela 4 – Taxa de aprovação sintática do agente, global e por variação de *prompt*

Grupo	N	APROVADO	REPROVADO	ERRO_EXECUCAO	Taxa de aprovação (%)
Global	150	75	75	0	50,0
Prompt 1	50	29	21	0	58,0
Prompt 2	50	25	25	0	50,0
Prompt 3	50	21	29	0	42,0

Fonte: Autoria própria (2026).

O laço de autocorreção entre os nós 4 e 5 foi acionado de forma intensa. Como mostra a Figura 4, a aprovação na primeira tentativa foi rara (13,3% das execuções), e a maioria (57,3%) consumiu as quatro tentativas disponíveis. O dado mais revelador está no destino dessas execuções que esgotaram o orçamento: das 86 que chegaram à quarta tentativa, apenas 11 foram aprovadas, enquanto 75 terminaram em falha. Ou seja, quando o erro não é corrigido nas primeiras iterações, tentativas adicionais raramente o resolvem.

Figura 4 – Distribuição do número de tentativas até a aprovação no nó de validação determinística.



Fonte: Autoria própria (2026).

A natureza desses erros é detalhada na Tabela 5. As falhas se concentram no plano da formatação: quase metade das reprovações decorre de YAML mal-formado, e, somadas as demais categorias de natureza estrutural (campos obrigatórios ausentes e *tags* MITRE inválidas),

a barreira sintática responde pela maioria expressiva das falhas. As não conformidades detectadas pelo *pySigma* (de natureza mais semântica, ligadas à especificação da linguagem) são minoritárias. Esse perfil localiza a fragilidade do agente na produção de YAML bem formado, e não na lógica da regra em si, resultado que reaparece de forma convergente nas discussões da Seção 5.2.

Tabela 5 – Distribuição dos erros de validação por categoria

Categoria do erro	Quantidade	Percentual (%)
Parsing YAML	34	45,3
Validação pelo *pySigma*	21	28,0
Campo obrigatório ausente	14	18,7
Tag MITRE inválida	6	8,0

Fonte: Autoria própria (2026).

Por fim, se examinou a influência das decisões dos nós iniciais sobre o desfecho da validação, a partir das variáveis `usou_web_search`, `tipo_input` e `contexto_pobre`. A primeira foi constante (`True`) nas 150 execuções e, portanto, não discrimina resultados. As outras duas revelam dois padrões dignos de nota (Tabela 6).

Um deles é que entradas classificadas como CVE apresentaram taxa de aprovação menor que as de texto livre. Uma hipótese é de que a presença de um CVE induz o agente a buscar e incorporar mais contexto técnico, o que resulta em regras mais complexas e mais suscetíveis a erro sintático.

O segundo padrão é contraintuitivo: cenários com contexto considerado pobre pelo nó 1 produziram aprovação mais alta que cenários com contexto considerado rico. A hipótese explicativa é a mesma que organiza toda esta seção: menos material de entrada leva o agente a gerar regras estruturalmente mais simples, mais fáceis de validar, ao passo que contextos ricos induzem tentativas mais elaboradas, que aumentam a exposição a erros de sintaxe. A validade formal, portanto, não cresce com a riqueza do contexto — em certa medida, decresce, porque sofisticação e correção sintática competem entre si nesta configuração do agente.

Tabela 6 – Resultados da validação por tipo de *input* e qualidade do contexto

Variável	Categoria	APROVADO	REPROVADO	Total	Taxa de aprovação (%)
`contexto_pobre`	Falso	47	54	101	46,5
	Verdadeiro	28	21	49	57,1
`tipo_input`	CVE	4	8	12	33,3
	Texto livre	71	67	138	51,4
Total Geral	—	**75**	**75**	**150**	**50,0**

Fonte: Autoria própria (2026).

5.1.2 Qualidade semântica das regras do agente

Cada regra gerada foi submetida à avaliação semântica do juiz do nó 6, que emite um de três vereditos: *APPROVED*, *APPROVED_WITH_WARNINGS* ou *REJECTED*. A Tabela 7 apresenta a distribuição desses vereditos nas 150 execuções, global e por variação de *prompt*.

O dado mais expressivo é a escassez de aprovações plenas: conforme a Tabela 7, menos de um décimo das regras recebeu *APPROVED* sem ressalvas, enquanto a grande maioria acumulou ao menos uma advertência. A leitura por *prompt* confirma a tendência já observada no desempenho determinístico (5.1.1), porém em sentido complementar: o *prompt3*, mais exigente e de menor aprovação sintática, é também o que produziu mais aprovações plenas e mais rejeições, concentrando os dois extremos. O *prompt1*, mais estável na sintaxe, quase não gera nem aprovações plenas nem rejeições, acumulando-se na faixa intermediária de ressalvas. Vê-se que *prompts* mais elaborados ampliam a variância da qualidade em ambas as pontas, e não deslocam uniformemente o desempenho para cima ou para baixo.

Tabela 7 – Distribuição dos vereditos emitidos pelo nó 6 (juiz semântico)

Grupo	N	APPROVED	APPROVED_WITH_WARNINGS	REJECTED	Não avaliado
Global	150	12	95	40	3
Prompt 1	50	2	46	1	1
Prompt 2	50	4	26	18	2
Prompt 3	50	6	23	21	0

Fonte: Autoria própria (2026).

O parecer do juiz se decompõe em seis dimensões independentes, cada uma classificada como *PASS*, *WARN* ou *FAIL*. A Tabela 8 apresenta a distribuição de cada status nas 147 regras efetivamente avaliadas, e é nela que se revela o principal achado da avaliação semântica: existe uma separação nítida entre as dimensões que medem a *forma* da regra e as que medem seu *conteúdo*.

As dimensões ligadas à engenharia formal concentram os melhores resultados. `invented_filters`, que verifica se o agente evitou introduzir filtros sem respaldo no cenário, e `condition`, que checa a coerência da lógica *booleana*, superam ambas os 78% de *PASS*. Na outra ponta estão as dimensões que dependem de conhecimento de domínio: `modifiers` e `logsource` ficam próximas ou abaixo da metade, e `semantic_alignment`, que mede se a lógica de detecção efetivamente captura a ameaça descrita, registra o pior desempenho de todas, com menos de um terço de *PASS*. Estes dados sugerem que o agente é competente na montagem formal da regra e frágil na tarefa de decidir o que a regra deve procurar para detectar a ameaça.

Tabela 8 – Resultado dos vereditos nas seis dimensões de avaliação do juiz semântico

Dimensão	N avaliado	PASS	WARN	FAIL	PASS (%)	WARN (%)	FAIL (%)
Filtros inventados	147	119	8	20	81,0	5,4	13,6
Condição	147	116	15	16	78,9	10,2	10,9
Estrutura	147	88	56	3	59,9	38,1	2,0
Logsource	147	74	33	40	50,3	22,4	27,2
Modificadores	147	64	78	5	43,5	53,1	3,4
Alinhamento semântico	147	43	62	42	29,3	42,2	28,6

Fonte: Autoria própria (2026).

Essa dissociação se torna mais evidente ao cruzar o veredito determinístico do nó 5 com o veredito semântico do nó 6, como mostra a Tabela 9. Das 75 regras aprovadas pelo validador sintático (YAML válido, em conformidade com a especificação *Sigma*, campos obrigatórios e *tags* MITRE corretos), apenas 7 receberam *APPROVED* do juiz. As demais 68 acumularam ao menos uma ressalva, e 21 foram *REJECTED* apesar da validade sintática. O caminho inverso também ocorre, ainda que em menor escala: 5 regras reprovadas pelo nó 5 receberam *APPROVED* do nó 6. A conformidade formal, portanto, não garante adequação semântica, nem a inadequação formal a exclui, o que estabelece que os dois validadores medem qualidades distintas e não substituíveis.

Tabela 9 – Cruzamento entre o status do nó 5 (validação determinística) e o veredito do nó 6 (juiz semântico)

Status sintático	APPROVED	APPROVED_WITH_WARNINGS	REJECTED	Não avaliado	Total
Aprovado	7	45	21	2	75
Reprovado	5	50	19	1	75
Total Geral	**12**	**95**	**40**	**3**	**150**

Fonte: Autoria própria (2026).

A análise qualitativa dos comentários textuais do juiz corrobora esse quadro. Como mostra a Tabela 10, as dimensões que concentram o maior volume de observações são justamente `semantic_alignment` e `logsource`, as mesmas com menor taxa de *PASS*. Comentários representativos apontam lógicas de detecção que não cobrem o escopo da ameaça descrita e fontes de `log` incorretamente atribuídos — por exemplo, uma regra classificada como *FAIL* em `semantic_alignment` por concentrar-se em palavras-chave isoladas e ignorar atividades mais amplas de reconhecimento e exfiltração, ou um `logsource` atribuído a `registry_event` quando o correto seria `database` [2]. A convergência entre a contagem de *PASS* por

[2] Uma amostra dos comentários, organizada por dimensão e status, está disponível em: www.github.com/daninot/agente-autonomo-rag-regras-sigma/blob/main/data/analises/amostras_comentarios.csv.

dimensão (Tabela 10) e o volume de comentários por dimensão reforça o diagnóstico de que os dois recortes, quantitativo e textual, apontam para a mesma fragilidade semântica.

Tabela 10 – Volume de comentários textuais do juiz por dimensão, segmentado por status

Dimensão	PASS	WARN	FAIL	Total de comentários
Alinhamento semântico	0	28	21	49
Logsource	0	23	19	42
Modificadores	0	38	3	41
Estrutura	0	29	1	30
Filtros inventados	0	7	13	20
Condição	0	10	9	19
Total		**135**	**66**	**201**

Fonte: Autoria própria (2026).

5.1.3 Similaridade com o gabarito

Esta seção descreve as duas métricas de proximidade empregadas para confrontar as regras geradas em cada um dos cinco grupos comparativos com as regras-gabarito do conjunto de teste: similaridade semântica de cosseno entre *embeddings*, calculada tanto sobre o documento YAML completo quanto exclusivamente sobre o bloco `detection` e correspondência estrutural em cinco campos categóricos da regra. Os *embeddings* foram produzidos pelo modelo `BAAI/bge-small-en-v1.5`.

Para fins de confirmação, o conjunto experimental são os cinco grupos de regras *Sigma* contra as 50 regras-gabarito do conjunto de teste. São eles: o E2, composto por 50 regras geradas em modo *zero-shot* por cinco modelos de linguagem comerciais; o E3, construído da mesma forma, porém com uso de engenharia de *prompt*; e os três subgrupos do E4, correspondentes às 50 regras geradas pelo agente proposto para cada uma das três variações de *prompt* avaliadas (E4-p1, E4-p2 e E4-p3).

5.1.3.1 Similaridade semântica

A Tabela 11 apresenta as estatísticas descritivas da similaridade de cosseno entre cada regra gerada e sua regra-gabarito, sob duas medidas: sobre o documento YAML completo e apenas sobre o bloco `detection`, o núcleo lógico da regra.

Na similaridade do YAML completo, os cinco grupos são próximos — todos com medianas acima de 0,80. Isso ocorre porque o YAML completo inclui campos de metadados (`title`, `description`, `tags`, `level`) cuja estrutura é semelhante entre regras, o que eleva a similaridade mesmo quando a lógica de detecção difere. A medida discriminante, portanto, é a do

bloco `detection`, e é nela que os grupos se separam: enquanto E2 e E3 mantêm medianas altas, os três subgrupos do E4 exibem desvios-padrão superiores a 0,39, cerca de oito vezes o do E3, sugerindo que a qualidade da `detection` gerada pelo agente varia intensamente de um cenário para outro.

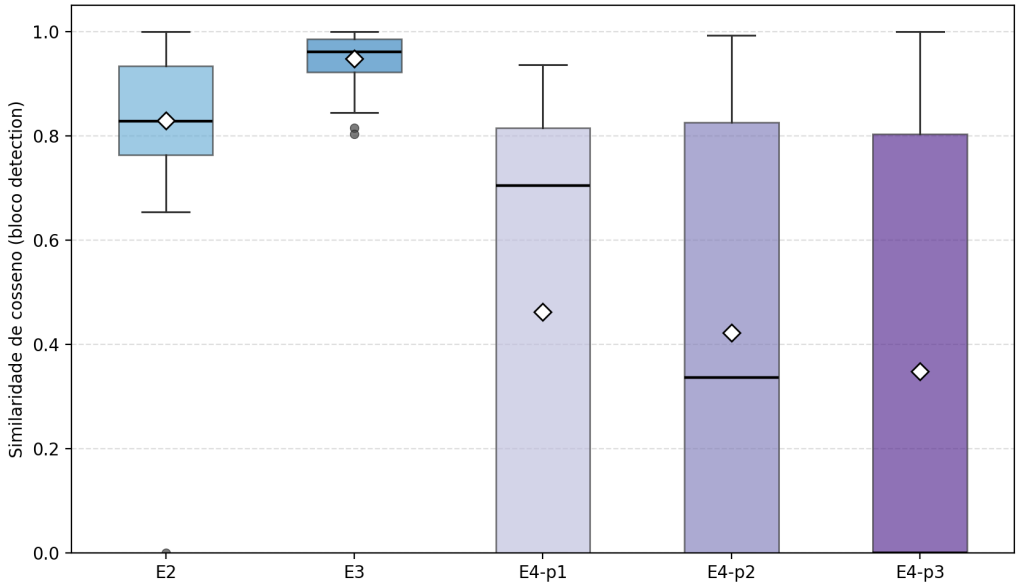
Tabela 11 – Resultados estatísticos de similaridade de cosseno global e de *detection* por grupo

	Similaridade Global			Similaridade *Detection*		
Grupo	**Média**	**Mediana**	**DP**	**Média**	**Mediana**	**DP**
E2	0,887	0,879	0,064	0,829	0,827	0,154
E3	0,927	0,931	0,030	0,948	0,961	0,048
E4-p1	0,805	0,812	0,063	0,462	0,704	0,398
E4-p2	0,853	0,842	0,046	0,421	0,336	0,425
E4-p3	0,866	0,856	0,055	0,347	0,000	0,413

Fonte: Autoria própria (2026).

Essa variância não se distribui de forma contínua, pois como ilustra a Figura 5, a similaridade da `detection` se concentra nos extremos: em todos os cinco grupos, a faixa intermediária (entre 0,1 e 0,5) está praticamente vazia. Em E2 e E3, quase todos os cenários caem na faixa alta; nos subgrupos do E4, a divisão é aproximadamente meio a meio entre a faixa alta e a baixa, acompanhando as próprias taxas de aprovação sintática de cada *prompt*. O agente, em regime autônomo, tende a produzir ou uma `detection` próxima do gabarito ou uma saída sem correspondência, mas raramente um resultado parcial.

Figura 5 – Distribuição da similaridade de cosseno entre os blocos `detection` das regras geradas e das regras-gabarito, por grupo.



Fonte: Autoria própria (2026).

A relação entre essa bimodalidade e a barreira sintática se confirma ao recalcular a similaridade de `detection` apenas sobre as regras com YAML válido, na Tabela 12. Sob essa restrição, as medianas dos três subgrupos do E4 sobem para a faixa de 0,80 e seus desvios-padrão caem para próximo de 0,08, valores comparáveis aos do E2. Ou seja: as saídas de baixa similaridade que puxavam a distribuição do E4 para baixo eram, em grande parte, regras com YAML inválido, cujo bloco `detection` sequer podia ser extraído. Quando o agente entrega uma regra sintaticamente válida, a qualidade semântica de sua `detection` é razoável, no entanto, sua fragilidade certamente está localizada na travessia da barreira sintática, e não na composição lógica da regra em si.

Tabela 12 – Estatísticas descritivas da similaridade do bloco `detection`, considerando apenas cenários com YAML válido na regra gerada

Grupo	N (válidos)	Média	Mediana	DP
E2	49	0,846	0,830	0,100
E3	50	0,948	0,961	0,048
E4-p1	29	0,796	0,807	0,081
E4-p2	25	0,843	0,827	0,077
E4-p3	21	0,827	0,819	0,094

Fonte: Autoria própria (2026).

5.1.3.2 Correspondência estrutural

A Tabela 13 apresenta a correspondência entre cada regra gerada e sua regra-gabarito em cinco campos categóricos: os três níveis de `logsource` (`category`, `product` e `service`) e o campo `level`, por acerto exato, e as *tags* MITRE, por índice médio de *Jaccard*. O bloco `detection` foi excluído desta análise: sua estrutura aninhada e a possibilidade de expressar a mesma intenção lógica por diferentes combinações de campos tornam o acerto exato inadequado, além de que sua avaliação foi feita por *embeddings* na subseção anterior. Regras com YAML inválido foram contabilizadas como erro em todos os campos.

O contraste entre grupos é acentuado. E3 acerta os três campos de `logsource` em 100% dos cenários e E2 fica na faixa de 53% a 70%; os três subgrupos do E4 não ultrapassam 20% em `category`, o campo mais determinante, pois define a natureza da fonte de *log*. O mesmo padrão se repete em `tags` e `level`. Parte dessa distância decorre da barreira sintática já discutida, uma vez que aqui as regras inválidas contam como erro.

Tabela 13 – Correspondência estrutural entre regras geradas e regras-gabarito, calculada sobre o total de 50 regras por grupo.

		Logsource **(%)**				
Grupo	**N considerado**	**Categoria**	**Produto**	**Serviço**	***Tags*** **(Jaccard)**	**Nível (%)**
E2	50	58,3	70,5	53,3	0,314	58,0
E3	50	100,0	100,0	100,0	0,333	54,0
E4-p1	50	5,6	27,3	20,0	0,067	12,0
E4-p2	50	19,4	40,9	26,7	0,108	28,0
E4-p3	50	16,7	25,0	13,3	0,062	16,0

Fonte: Autoria própria (2026).

Para isolar esse efeito, a Tabela 14 recalcula a correspondência apenas sobre as regras com YAML válido. A melhora do E4 é parcial: em `category`, os subgrupos sobem para a faixa de 10% a 50%, e em `product` chegam a ultrapassar 80% em um dos subgrupos. Ainda assim, permanecem consistentemente abaixo do E2 e muito distantes do E3, que mantém 100%.

Tabela 14 – Correspondência estrutural restrita às regras com YAML válido por grupo.

		Logsource **(%)**				
Grupo	**N considerado**	**Categoria**	**Produto**	**Serviço**	***Tags*** **(Jaccard)**	**Nível (%)**
E2	49	60,0	72,1	53,3	0,320	59,2
E3	50	100,0	100,0	100,0	0,333	54,0
E4-p1	29	10,5	50,0	27,3	0,115	20,7
E4-p2	25	41,2	81,8	50,0	0,216	56,0
E4-p3	21	50,0	61,1	22,2	0,148	38,1

Fonte: Autoria própria (2026).

Essa persistência da distância, mesmo após a remoção das regras sintaticamente inválidas, distingue a correspondência estrutural da similaridade de `detection` analisada anteriormente. Enquanto esta última filtrada aproximava o E4 dos modelos comerciais, o acerto de `logsource` não se recupera da mesma forma: o agente erra a fonte de `log` adequada mesmo quando produz uma regra formalmente correta. Trata-se, portanto, de uma limitação semântica, e não sintática, coerente com o baixo desempenho da dimensão `logsource` no juiz (5.1.2). Uma hipótese explicativa remete à etapa de recuperação, no RAG: quando a recuperação de contexto traz exemplos de categorias de *log* heterogêneas, o gerador não recebe um sinal consistente sobre qual fonte adotar, e o diagnóstico de coerência embutido naquele nó registra justamente essa dispersão.

Por fim, a Tabela 15 consolida a taxa de YAML válido por grupo, que define os denominadores das análises filtradas (Tabelas 12 e 14). A distância entre os quase 100% de E2 e E3 e

os 42% a 58% do E4 quantifica, em um único indicador, a barreira sintática que perpassa todos os resultados do agente.

Tabela 15 – Taxa de regras com YAML válido por grupo.

Grupo	N	YAML válidos	Taxa (%)
E2	50	49	98,0
E3	50	50	100,0
E4-p1	50	29	58,0
E4-p2	50	25	50,0
E4-p3	50	21	42,0

Fonte: Autoria própria (2026).

5.2 Discussões

As discussões a seguir se organizam em torno de quatro questões que emergem da leitura dos três instrumentos de avaliação empregados: o validador determinístico (Seção 5.1.1), o juiz semântico (Seção 5.1.2) e as métricas externas de similaridade e correspondência (Seção 5.1.3). A primeira examina a relação entre a competência formal e a competência semântica do agente. A segunda localiza dentro dessa relação uma barreira específica: a validade sintática do YAML. A terceira examina o que as três variações de *prompt* revelam sobre os regimes de operação do agente e a quarta calibra o alcance das conclusões possíveis a partir da comparação com abordagens baseadas em LLMs comerciais.

5.2.1 Dissociação entre forma e conteúdo

O principal achado deste trabalho é o descolamento entre o desempenho do agente na construção formal da regra e o desempenho no alinhamento semântico entre a lógica de detecção produzida e a intenção do cenário.

No plano formal, três indicadores convergem. A dimensão de condição do juiz semântico registrou 78,9% de *PASS*, e a dimensão de filtros inventados, 81,0%, o que indica composição consistente de estruturas *booleanas* e baixa incidência de campos fora do padrão *Sigma*. A similaridade global média entre as regras do E4 e as regras-gabarito ficou entre 0,805 e 0,866 (Tabela 11), próxima das faixas observadas em E2 (0,887) e E3 (0,927), o que sugere preservação da estrutura textual da regra — *title*, *description*, *logsource*, *tags*, *level*, *detection* e *condition*.

Já no plano semântico, o quadro difere. A dimensão de alinhamento semântico do juiz atingiu 29,3% de *PASS*, valor inferior ao das demais dimensões avaliadas. A correspondência exata em *logsource.category* ficou entre 5,6% e 19,4% nos três subgrupos do E4, contra 100%

no E3 e 58,3% no E2. Os índices de *Jaccard* sobre *tags* variaram entre 0,062 e 0,108. As três métricas, embora não sejam estatisticamente independentes, apontam para a fragilidade da definição da lógica de detecção adequada ao cenário descrito.

A separação entre o nó 5 (validador sintático) e o nó 6 (juiz semântico) permite dimensionar esse descolamento. Das 75 regras aprovadas pelo validador, apenas 7 (9,3%) receberam veredito *APPROVED* do juiz, e 21 (28,0%) foram classificadas como *REJECTED* apesar da validade sintática. No sentido inverso, 5 regras reprovadas pelo validador receberam *APPROVED* do juiz. A validade formal, portanto, não implica adequação semântica, e a adequação semântica não pressupõe validade formal. A verificação sintática captura apenas parte da qualidade operacional da saída. Este descolamento sustenta o agente apenas como ferramenta de apoio ao especialista, já que a revisão humana antes da implantação incide sobre uma dimensão que os filtros formais não cobrem.

5.2.2 A barreira sintática como gargalo principal

A fragilidade formal do agente se concentra na produção de YAML sintaticamente válido. A Tabela 15 registra taxas de YAML válido entre 42% e 58% nos três subgrupos do E4, contra 98% no E2 e 100% no E3. Sobre o conjunto completo de 50 regras, as medianas da similaridade de `detection` nos subgrupos do E4 foram 0,704, 0,336 e 0,000 (Tabela 11). Restringindo a análise às regras com YAML válido (Tabela 12), as medianas passam a 0,807, 0,827 e 0,819, próximas ao 0,830 do E2. Sob a condição de validade formal, o desempenho do agente na composição semântica de `detection` se aproxima às das saídas comerciais filtradas.

O padrão de distribuição confirma essa leitura. Como ilustra a Figura 5, a similaridade de `detection` se concentra nos extremos: a faixa intermediária (entre 0,1 e 0,5) permanece praticamente vazia nos cinco grupos, e as regras se distribuem entre a faixa alta ($> 0{,}5$) e a baixa ($\leq 0{,}1$), sem gradiente entre elas. O agente opera, assim, em dois regimes: ou entrega uma saída sintaticamente válida com `detection` aceitável, ou uma saída sintaticamente inviável. A ausência de um regime intermediário implica no custo de correção de uma saída inválida, que aproxima-se do custo de escrever uma regra nova.

Essa leitura, contudo, não estabelece equivalência entre E4 e E2. O E2 é composto por regras selecionadas, de validade formal asseguradas; o E4, por sua vez, apresenta entre 42% e 58% de saídas inválidas. O que os dados sustentam é uma afirmação condicional: quando o agente entrega YAML válido, a qualidade de `detection` tange à das saídas comerciais filtradas, o que reposiciona a barreira sintática, e não a composição lógica da regra, como o gargalo prioritário a ser enfrentado.

5.2.3 Variações de *prompt*

Os três subgrupos do E4 permitem examinar se alguma variação de *prompt* configura desempenho superior. Os dados indicam que as três correspondem a regimes distintos, cuja seleção depende do contexto de uso.

Na validação sintática, o ordenamento é p1 (58,0%) $>$ p2 (50,0%) $>$ p3 (42,0%); já na avaliação semântica, ele se inverte: p1 produziu 2 regras *APPROVED* contra 6 de p3, que também concentrou o maior número de *REJECTED*. Das 50 regras de p1, 46 receberam veredito intermediário, enquanto p3 concentra suas saídas nos extremos. O padrão sugere que p1, por solicitar saídas menos elaboradas, induz respostas conservadoras, que atendem à validação com mais frequência mas raramente atingem qualidade plena; p3 solicita saídas mais elaboradas e amplia a variância, aumentando ao mesmo tempo os acertos plenos e as falhas completas. O tamanho amostral (50 regras por subgrupo) não permite inferência estatística firme, mas a consistência do padrão entre métricas independentes reduz a probabilidade de ruído.

A escolha entre variações acaba dependendo da decisão de calibração conforme o cenário. Em *pipelines* automatizados de SOC, nos quais a validade formal condiciona o processamento subsequente, p1 é a opção mais estável; em uso assistido por analista, no qual saídas de alta qualidade compensam o descarte de saídas inviáveis, p3 é mais adequado.

5.2.4 Comparação com LLMs comerciais

A comparação entre o agente e os grupos comerciais sustenta a hipótese central do trabalho, mas levanta três diferenças estruturais que condicionam as possíveis conclusões.

A primeira trata sobre o viés de filtragem, pois os grupos E2 e E3 tiveram suas melhores regras selecionadas por uma etapa de filtragem, com LLMs atuando como juiz, procedimento que favorece saídas alinhadas ao que LLMs produzem consistentemente (Zheng *et al.*, 2023), enquanto que o E4 não passa por filtragem análoga. A segunda é a escala de infraestrutura, já que as cinco LLMs dos grupos E2 e E3 (*ChatGPT*, *Claude*, *DeepSeek*, *Gemini* e *Microsoft Copilot*) têm número de parâmetros muito superior ao do *Llama 3.1 8B*, pois passaram por ajuste extenso através de *feedback* humano e acessam a internet durante a inferência, vantagem que se manifesta em tarefas de aderência a gramáticas rígidas como o YAML. A terceira é a provável exposição prévia ao domínio: o repositório *SigmaHQ* é público e amplamente disseminado, sendo plausível que parte de seu conteúdo esteja nos dados de treinamento das LLMs comerciais, o que pode explicar a correspondência de 100% do E3 em *logsource*; o E4 acessa o mesmo conteúdo por RAG, e não como conhecimento internalizado.

Consideradas as três diferenças, a conclusão é que o agente, na configuração atual, não supera abordagens comerciais filtradas em operação totalmente autônoma, e parte relevante dessa diferença decorre de fatores externos à sua arquitetura. A conclusão é compatível com o posicionamento do agente como ferramenta de apoio ao especialista, no qual o critério deixa

de ser desempenho autônomo em *benchmark* e passa a ser utilidade em fluxo de trabalho assistido.

5.3 Limitações e Trabalhos futuros

Os resultados apresentados neste trabalho precisam ser interpretados considerando algumas limitações. A primeira é o tamanho da amostra; o conjunto de teste contém 50 cenários, número reduzido para sustentar conclusões estatísticas firmes sobre as diferenças observadas entre os grupos comparativos. Afirmações definitivas sobre a superioridade de um método sobre outro exigem uma amostra maior. As demais limitações estão ligadas à construção do artefato. O agente foi desenhado para execução local, sob restrições de *hardware* doméstico, o que levou à escolha do *Llama 3.1 8B* como modelo gerador principal, modelo bem menor que as LLMs comerciais usadas nos grupos E2 e E3. Parte da diferença de desempenho observada, portanto, vem dessa diferença de escala e não da arquitetura do agente em si. Além disso, o modelo foi usado em sua versão pré-treinada, sem nenhum ajuste específico para a geração de regras *Sigma*; técnicas de treinamento supervisionado poderiam aproximá-lo da estrutura formal da linguagem e reduzir a fração de saídas com YAML inválido observada no E4 (entre 42% e 58% conforme a variação de *prompt*).

O processo de avaliação também tem limitações que precisam ser registradas. A montagem dos grupos E2 e E3, que envolveu selecionar as melhores regras geradas por LLMs comerciais, favoreceu saídas alinhadas com a regra-gabarito — exatamente a propriedade que as métricas externas medem. A comparação direta entre o agente e as abordagens comerciais fica, por isso, desequilibrada. Além disso, a avaliação semântica feita pelo nó 6 e pelos cinco juízes da filtragem foi conduzida inteiramente por modelos de linguagem. A literatura mostra boa correlação entre *LLM-as-a-judge* e avaliação humana em tarefas estruturadas (Zheng *et al.*, 2023), mas essa correspondência não é total, e nuances de qualidade de uma regra *Sigma* podem escapar a um avaliador automático, o que reafirma a importância da avaliação humana no estudo.

A partir dessas limitações, este trabalho aponta cinco direções concretas para pesquisa futura. A primeira corresponde à limitação do tamanho amostral: replicar o experimento com 100 ou mais cenários, preferencialmente cobrindo diferentes famílias de *logsource* de forma equilibrada, o que permitiria inferência estatística mais firme e análises estratificadas por tipo de cenário. A segunda, à limitação de escala do modelo: reproduzir o experimento substituindo o *Llama 3.1 8B* por modelos abertos maiores (como variantes do *Llama 3.1 70B*), executados em infraestrutura adequada. Esse experimento isolaria o efeito da escala do modelo sobre o desempenho do agente. A terceira sugestão diz respeito à ausência de treinamento especializado: aplicar *fine-tuning* supervisionado ao modelo gerador via técnicas como LoRA (Hu *et al.*, 2021), usando pares de regra *Sigma* e instrução em linguagem natural. Essa abordagem exige a cons-

trução de um *dataset* adequado e validação cuidadosa contra superajuste, mas tem potencial para reduzir a fração de saídas com YAML inválido observada no E4.

As duas últimas sugestões tratam do procedimento de avaliação e à arquitetura do próprio agente. A primeira delas é a validação com avaliação humana: submeter uma amostra estratificada de regras à análise de *detection engineers* experientes, segundo uma rubrica definida *a priori*, o que permitiria calibrar o quanto a avaliação semântica automatizada deste trabalho corresponde ao julgamento de profissionais da área. A segunda é o reposicionamento do juiz semântico como bloqueador no fluxo do agente. Na arquitetura atual, o juiz semântico (nó 6) atua apenas como avaliador pós-execução, ou seja, emite parecer sobre a regra finalizada, mas não interfere no fluxo de geração. Os resultados da Seção 5.1.1 mostraram que, das 75 regras aprovadas pelo validador sintático (nó 5), apenas 9,3% receberam veredito `APPROVED` do juiz e 28,0% foram `REJECTED` apesar da validade sintática. Isso significa que parte das regras formalmente válidas chega ao usuário com problemas semânticos. Uma direção alternativa é reposicionar o juiz como segundo bloqueador no laço de autocorreção: regras `REJECTED` retornariam ao nó gerador junto com as justificativas por dimensão; regras `APPROVED_WITH_WARNINGS` seriam devolvidas para refinamento, sob limite máximo de tentativas. A modificação transformaria o juiz, de instrumento de avaliação, em instrumento de controle de qualidade da saída.

6 CONCLUSÃO

Este trabalho partiu da constatação de que a velocidade da produção de regras de detecção se tornou um fator estratégico em cibersegurança, diante do crescimento exponencial de vulnerabilidades e da compressão do intervalo entre divulgação e exploração das mesmas. Em resposta a essa lacuna, foi proposto e implementado um agente autônomo capaz de gerar regras *Sigma* a partir de fontes diversificadas de inteligência de ameaças, apoiado por geração aumentada por recuperação (RAG) sobre o repositório *SigmaHQ* e por validação iterativa da saída.

Os dados obtidos em 150 execuções sobre 50 cenários distintos sustentam uma conclusão restrita: o agente cumpre seu papel exclusivamente como ferramenta de apoio ao analista, não como substituto. A taxa global de aprovação sintática de 50,0% indica que metade das saídas produzidas em regime autônomo não é utilizável sem intervenção humana, e a fragilidade se concentra em duas frentes distintas. A primeira é sintática: sem o auxílio de mecanismos de geração restrita por gramática, o modelo tem dificuldade em produzir YAML rigorosamente válido, taxa muito inferior às de 98,0% e 100,0% observadas nos grupos comparativos baseados em LLMs comerciais. A segunda, mais sutil e revelada apenas pela leitura combinada das métricas de avaliação, é de que o agente demonstra bom desempenho na construção da forma da regra quanto à estrutura, campos obrigatórios, metadados e lógica booleana bem formada, com 78,9% de *PASS* na dimensão de condição e 81,0% na dimensão de filtros inventados, mas apresenta desempenho substancialmente inferior no alinhamento semântico com a intenção do cenário, dimensão que atingiu apenas 29,3% de *PASS*. Em outras palavras, o agente é competente na engenharia formal da regra *Sigma*, mas frágil na tarefa de decidir o que a regra deve efetivamente procurar para detectar a ameaça descrita. Esse conjunto de resultados evidencia que, em regime autônomo, o desempenho do agente não supera o das LLMs comerciais empregadas de forma direta, limitação que reflete tanto a fragilidade sintática quanto a assimetria entre forma e conteúdo observada na avaliação semântica. A revisão por analista experiente antes da implantação operacional, portanto, não é apenas recomendável como é condição ideal.

Para além do produto da pesquisa, este trabalho contribui com três seguimentos metodológicos que podem ser reaproveitados em pesquisas futuras. O primeiro é a metodologia comparativa organizada em quatro estágios de complexidade progressiva. As regras originais de especialistas, geração *zero-shot* com modelo comercial, engenharia de *prompt* e agente autônomo com RAG, que permite isolar o ganho marginal de cada camada arquitetural. Em seguida, o conjunto de instrumentos de avaliação que distingue explicitamente três dimensões complementares: a validação determinística da forma, a similaridade textual com regras de referência via *embeddings* e a avaliação semântica por juiz baseado em LLM decomposta em seis dimensões independentes. A leitura articulada dessas três dimensões, e não de qualquer uma isolada, é o que permite diagnosticar onde reside a fragilidade do agente. E por fim, o uso combi-

nado das métricas de similaridade semântica e correspondência estrutural entre regras geradas e regras-gabarito do *SigmaHQ*, que pode ser usado para replicar estudos semelhantes.

O trabalho aponta direções bem definidas para a continuidade da pesquisa. A primeira é o *fine-tuning* supervisionado do modelo gerador com pares de instrução e regra *Sigma*, abordagem que pode mitigar tanto a barreira sintática quanto a lacuna de alinhamento semântico identificadas nos resultados, ao expor o modelo à correspondência entre descrições textuais de ameaças e as lógicas de detecção efetivas encontradas em regras humanas de alta qualidade. A segunda é o reposicionamento do juiz semântico como bloqueador no fluxo de autocorreção, transformando-o de instrumento de avaliação pós-execução em mecanismo de controle de qualidade da saída, onde regras rejeitadas retornariam ao nó gerador acompanhadas das justificativas por dimensão, sob limite máximo de tentativas, com destaque para o retorno específico das falhas em alinhamento semântico, que é a dimensão com maior potencial de ganho. Por fim, a validação com avaliação humana, por meio da submissão de uma amostra estratificada de regras à análise de *detection engineers* experientes, o que permitiria calibrar o quanto a avaliação semântica automatizada corresponde ao julgamento profissional. Enquanto tais direções não são exploradas, a utilidade prática do agente permanece condicionada à articulação com o especialista.

REFERÊNCIAS

Accenture. **Securing the Digital Economy: Reinventing the Internet for Trust**. Dublin, Ireland, 2019. Relatório corporativo. Disponível em: https://shre.ink/3UM4.

ALCANTARA, L. *et al.* Syrius: Synthesis of rules for intrusion detectors. **IEEE Transactions on Reliability**, IEEE, v. 71, n. 1, p. 370–381, 2021.

BANERJEE, S.; LAVIE, A. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. *In*: **Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization**. [*S.l.*: *s.n.*], 2005. p. 65–72.

BOOTH, H.; RIKE, D.; WITTE, G. A. **The National Vulnerability Database (NVD): Overview**. Gaithersburg, MD, 2013.

BROWN, T. B. *et al.* Language models are few-shot learners. **Advances in Neural Information Processing Systems**, v. 33, p. 1877–1901, 2020.

BUSINESS, V. **2018 Data Breach Investigations Report**. New York, USA, 2018. Disponível em: https://www.verizon.com/business/resources/reports/DBIR_2018_Report.pdf.

BUSINESS, V. **2025 Data Breach Investigations Report (DBIR)**. USA, 2025. C. David Hylender, Philippe Langlois, Alex Pinto, Suzanne Widup (eds.). Disponível em: https://www.verizon.com/business/resources/T437/reports/2025-dbir-data-breach-investigations-report.pdf.

CASELLI, T. *et al.* When it's all piling up: Investigating error propagation in an NLP pipeline. *In*: **Proceedings of the 1st Workshop on NLP Applications: Completing the Puzzle (WNACP 2015)**. [*S.l.*: *s.n.*], 2015.

CHOWDHARY, K. R. Natural language processing. *In*: **Fundamentals of Artificial Intelligence**. New Delhi: Springer, 2020. p. 603–649.

DIMITROV, D.; NIKOLOV, D. Leveraging yara and sigma rules to detect chinese state-sponsored hacking groups of the "typhoon" type. *In*: **Environment. Technology. Resources. Proceedings of the 16th International Scientific and Practical Conference**. Rezekne, Latvia: RTU Press, 2025. II, p. 107–113. Disponível em: https://doi.org/10.17770/etr2025vol2.8617.

DU, M.; HU, W.; HEWLETT, W. AutoCombo: Automatic malware signature generation through combination rule mining. *In*: **Proceedings of the 30th ACM International Conference on Information & Knowledge Management**. [*S.l.*]: ACM, 2021. p. 3777–3786.

EXABEAM. **AI SIEM: How SIEM with AI/ML is Revolutionizing the SOC**. 2025. Disponível em: https://www.exabeam.com/explainers/siem/ai-siem-how-siem-with-ai-ml-is-revolutionizing-the-soc/. Acesso em: 20 jun. 2025.

Forum of Incident Response and Security Teams. **Common Vulnerability Scoring System version 3.1: Specification Document**. [*S.l.*], 2019. Disponível em: https://www.first.org/cvss/v3-1/specification-document.

GAMBLIN, J. **2025 CVE Data Review**. 2026. Análise consolidada dos dados da National Vulnerability Database (NVD): 48.185 CVEs publicados em 2025. Disponível em: https://jerrygamblin.com/2026/01/01/2025-cve-data-review/.

GONZÁLEZ-GRANADILLO, G.; GONZÁLEZ-ZARZOSA, S.; DIAZ, R. Security information and event management (SIEM): Analysis, trends, and usage in critical infrastructures. **Sensors**, v. 21, n. 14, p. 4759, 2021.

HASANOV, I. *et al.* Application of large language models in cybersecurity: A systematic literature review. **IEEE Access**, IEEE, v. 12, p. 176751–176778, 2024.

HU, E. J. *et al.* LoRA: Low-rank adaptation of large language models. **arXiv preprint arXiv:2106.09685**, 2021. Apresentado na International Conference on Learning Representations (ICLR), 2022. Disponível em: https://arxiv.org/abs/2106.09685.

IBM Security. **Cost of a Data Breach Report 2024**. Armonk, NY, 2024. Pesquisa conduzida pelo Ponemon Institute. Disponível em: https://www.ibm.com/downloads/documents/us-en/107a02e94948f4ec.

IBM Security. **Cost of a Data Breach Report 2025**. Armonk, NY, 2025. Pesquisa conduzida pelo Ponemon Institute. Disponível em: https://www.ibm.com/reports/data-breach.

JACCARD, P. The distribution of the flora in the alpine zone. **New Phytologist**, Wiley, v. 11, n. 2, p. 37–50, 1912.

JOHNSON, J.; DOUZE, M.; JÉGOU, H. Billion-scale similarity search with GPUs. **IEEE Transactions on Big Data**, IEEE, v. 7, n. 3, p. 535–547, 2019.

JORDAN, M. I.; MITCHELL, T. M. Machine learning: Trends, perspectives, and prospects. **Science**, American Association for the Advancement of Science, v. 349, n. 6245, p. 255–260, 2015.

JURAFSKY, D.; MARTIN, J. H. **Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition**. 2. ed. Upper Saddle River, NJ: Pearson Prentice Hall, 2009. ISBN 9780131873216.

KAUFMAN, S. *et al.* Leakage in data mining: Formulation, detection, and avoidance. **ACM Transactions on Knowledge Discovery from Data**, ACM, v. 6, n. 4, p. 1–21, 2012.

LangChain. **Graph API overview — LangGraph documentation**. 2024. Https://docs.langchain.com/oss/python/langgraph/graph-api. Acesso em: 17 jun. 2026.

LangChain Team. **LangGraph: Build resilient language agents as graphs**. 2024. Https://langchain-ai.github.io/langgraph/. Acesso em: 10 de junho.

LangChain Team. **LangGraph: Building Stateful, Multi-Actor Applications with LLMs**. 2024. Disponível em: https://langchain-ai.github.io/langgraph/.

LEMPINEN, M.; PYYNY, E.; JUNTUNEN, A. **Chatbot for Assessing System Security with OpenAI GPT-3.5**. 2023. 34 p.

LEWIS, P. *et al.* Retrieval-augmented generation for knowledge-intensive nlp tasks. *In*: **Advances in Neural Information Processing Systems**. [*S.l.*: *s.n.*], 2020. v. 33, p. 9459–9474.

LI, J. *et al.* **GRIDAI: Generating and Repairing Intrusion Detection Rules via Collaboration among Multiple LLM-based Agents**. 2025.

LIDDY, E. D. Natural language processing. *In*: **Encyclopedia of Library and Information Science**. 2. ed. New York: Marcel Decker, Inc., 2001.

LIN, C.-Y. ROUGE: A package for automatic evaluation of summaries. *In*: **Text Summarization Branches Out: Proceedings of the ACL-04 Workshop**. [*S.l.*: *s.n.*], 2004. p. 74–81.

LIU, P. *et al.* Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. **ACM Computing Surveys**, ACM, v. 55, n. 9, p. 1–35, 2023.

MADAAN, A. *et al.* Self-Refine: Iterative refinement with self-feedback. *In*: **Advances in Neural Information Processing Systems**. [*S.l.*: *s.n.*], 2023. v. 36.

Mandiant. Relatório técnico anual, **M-Trends 2026: Data, Insights, and Strategies From the Frontlines**. 2026. Tempo médio para exploração (time-to-exploit) estimado em -7 dias; exploits como vetor inicial mais comum pelo 6º ano consecutivo (32%). Disponível em: https://cloud.google.com/blog/topics/threat-intelligence/m-trends-2026.

MANNING, C. D.; RAGHAVAN, P.; SCHÜTZE, H. **Introduction to Information Retrieval**. Cambridge, UK: Cambridge University Press, 2008. ISBN 9780521865715.

MCGOWAN, M. **Detecting malicious activities with Sigma rules**. 2025. Disponível em: https://lantern.splunk.com/Security/UCE/Guided_Insights/Threat_hunting/Detecting_malicious_activities_with_Sigma_rules. Acesso em: 20 jun. 2025.

National Institute of Standards and Technology. **National Vulnerability Database – General Information**. 2025. Disponível em: https://nvd.nist.gov/general.

National Institute of Standards and Technology. **NIST Updates NVD Operations to Address Record CVE Growth**. 2026. Aumento de 263% nas submissões de CVE entre 2020 e 2025; adoção de modelo de priorização baseado em risco. Disponível em: https://www.nist.gov/news-events/news/2026/04/nist-updates-nvd-operations-address-record-cve-growth.

NOGUEIRA, R.; CHO, K. Passage re-ranking with BERT. **arXiv preprint arXiv:1901.04085**, 2019.

NURUSHEVA, A. *et al.* Machine learning algorithms in siem systems for enhanced detection and management of security events. *In*: **Bulletin of L. N. Gumilyov Eurasian National University. Mathematics, Computer Science, Mechanics Series**. Astana, Kazakhstan: L. N. Gumilyov Eurasian National University, 2024. v. 148, n. 3, p. 6–17.

PALO ALTO NETWORKS. **What Is the Role of AI and ML in Modern SIEM Solutions?** 2025. Disponível em: https://www.paloaltonetworks.com/cyberpedia/role-of-artificial-intelligence-ai-and-machine-learning-ml-in-siem. Acesso em: 20 jun. 2025.

PAPINENI, K. *et al.* BLEU: a method for automatic evaluation of machine translation. *In*: **Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)**. [*S.l.*: *s.n.*], 2002. p. 311–318.

PICUS SECURITY. **What Is a YARA Rule?** 2025. Disponível em: https://www.picussecurity.com/resource/glossary/what-is-a-yara-rule. Acesso em: 20 jun. 2025.

Picus Security. **The Top Ten MITRE ATT&CK Techniques**. 2026. Disponível em: https://www.picussecurity.com/resource/the-top-ten-mitre-attack-techniques.

PODZINS, O.; ROMANOVS, A. Why siem is irreplaceable in a secure it environment? *In*: **2019 Open Conference of Electrical, Electronic and Information Sciences (eStream)**. [*S.l.*: *s.n.*], 2019. p. 1–5.

RAJAPAKSHA, S.; RANI, R.; KARAFILI, E. A RAG-based question-answering solution for cyber-attack investigation and attribution. *In*: **Computer Security – ESORICS 2024**. [*S.l.*]: Springer, 2024. (Lecture Notes in Computer Science), p. 238–256.

Rapid7 Labs. Relatório técnico anual, **The 2026 Global Threat Landscape Report: Decoding the Accelerated Cyber Attack Cycle**. 2026. Mediana entre divulgação de vulnerabilidade e inclusão no catálogo CISA KEV reduzida de 8,5 para 5,0 dias; exploração confirmada de vulnerabilidades CVSS 7-10 cresceu 105% (de 71 em 2024 para 146 em 2025). Disponível em: https://www.rapid7.com/research/report/global-threat-landscape-report-2026/.

REIMERS, N.; GUREVYCH, I. Sentence-BERT: Sentence embeddings using siamese BERT-networks. *In*: **Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)**. Hong Kong, China: Association for Computational Linguistics, 2019. p. 3982–3992.

ROTH, F. **Rule Creation Guide**. 2022. Disponível em: https://github.com/SigmaHQ/sigma/wiki/Rule-Creation-Guide. Acesso em: 20 jun. 2025.

ROY, S. *et al.* **SoK: The MITRE ATT&CK Framework in Research and Practice**. 2023. arXiv:2304.07411 [cs.CR]. Disponível em: https://arxiv.org/abs/2304.07411.

RUSSELL, S.; NORVIG, P. **Artificial Intelligence: A Modern Approach**. 4. ed. Hoboken, NJ: Pearson, 2020. ISBN 9780134610993.

SADDI, V. R. *et al.* Examine the role of generative ai in enhancing threat intelligence and cyber security measures. *In*: **Proceedings of the 2024 2nd International Conference on Disruptive Technologies (ICDT)**. India: IEEE, 2024. p. 537–542.

SANS Institute. White paper, **SANS 2025 Detection and Response Survey**. 2025. Survey anual independente e vendor-neutral (3ª edição), conduzido por Josh Lemon. 73% das organizações apontam falsos positivos como o principal desafio de detecção; 76% citam fadiga de alertas como preocupação central do SOC. Disponível em: https://www.sans.org/white-papers/2025-detection-response-survey-webcast-forum.

SCHWARTZ, Y. *et al.* LLMCloudHunter: Harnessing LLMs for automated extraction of detection rules from cloud-based CTI. *In*: **Proceedings of the ACM Web Conference 2025 (WWW '25)**. Sydney, NSW, Australia: ACM, 2025. p. 1922–1941. ISBN 979-8-4007-1274-6/25/04.

SHAH, A. H. *et al.* Automated log analysis and anomaly detection using machine learning. *In*: **Fuzzy Systems and Data Mining VIII**. Amsterdam, Netherlands: IOS Press, 2022, (Frontiers in Artificial Intelligence and Applications, v. 358). p. 137–147.

SHEERAZ, M. *et al.* Revolutionizing SIEM security: An innovative correlation engine design for multi-layered attack detection. **Sensors**, v. 24, n. 15, p. 4901, 2024.

SHEPARDSON, D. **SMarriott CEO apologizes for data breach, unsure if China responsible**. 2019. Disponível em: https://www.reuters.com/article/us-usa-cyber-congress/marriott-ceo-apologizes-for-data-breach-unsure-if-china-responsible-idUSKCN1QO217/. Acesso em: 20 jun. 2025.

SIGMAHQ. **About Sigma**. 2025. Disponível em: https://sigmahq.io/docs/guide/about.html. Acesso em: 20 jun. 2025.

SIGMAHQ. **Sigma - Generic Signature Format for SIEM Systems**. 2025. Disponível em: https://github.com/SigmaHQ/sigma/tree/master. Acesso em: 20 jun. 2025.

SIGMAHQ. **Sigma Rules**. 2025. Disponível em: https://sigmahq.io/docs/basics/rules.html. Acesso em: 20 jun. 2025.

STROM, B. E. *et al.* **MITRE ATT&CK: Design and Philosophy**. [*S.l.*], 2018. Revised March 2020.

TARIQ, S. *et al.* Alert fatigue in security operations centres: Research challenges and opportunities. **ACM Computing Surveys**, v. 57, n. 9, p. 1–38, 2025.

TENDIKOV, N. *et al.* Security information event management data acquisition and analysis methods with machine learning principles. **Results in Engineering**, v. 22, p. 102254, 2024.

The MITRE Corporation. **CVE Program Celebrates 25 Years of Impact in Cybersecurity**. 2024. Disponível em: https://www.mitre.org/news-insights/news-release/cve-program-celebrates-25-years-impact-cybersecurity.

The MITRE Corporation. **About CWE – Common Weakness Enumeration Overview**. 2025. Disponível em: https://cwe.mitre.org/about/index.html.

The MITRE Corporation. **CVE List Basics: Identifier Syntax and Numbering Authorities**. 2025. Disponível em: https://www.cve.org/ResourcesSupport/AllResources/CNARules.

VASWANI, A. *et al.* Attention is all you need. **Advances in Neural Information Processing Systems**, v. 30, 2017.

VIELBERTH, M. *et al.* Security operations center: A systematic study and open challenges. **IEEE Access**, v. 8, p. 227756–227779, 2020.

WANG, H. *et al.* RulePilot: An LLM-powered agent for security rule generation. *In*: **Proceedings of the 2026 IEEE/ACM 48th International Conference on Software Engineering (ICSE '26)**. Rio de Janeiro, Brazil: ACM, 2026.

WANG, L. *et al.* A survey on large language model based autonomous agents. **Frontiers of Computer Science**, Springer, v. 18, n. 6, p. 186345, 2024.

Wazuh, Inc. **Wazuh - The Open Source Security Platform**. 2024. https://wazuh.com/. Copyright © 2015-2024 Wazuh, Inc.; Accessed: 26 June 2025.

WEI, J. *et al.* Chain-of-thought prompting elicits reasoning in large language models. *In*: **Advances in Neural Information Processing Systems**. [*S.l.*: *s.n.*], 2022. v. 35, p. 24824–24837.

XIAO, S. *et al.* C-Pack: Packaged resources to advance general Chinese embedding. **arXiv preprint arXiv:2309.07597**, 2023.

YAO, S. *et al.* ReAct: Synergizing reasoning and acting in language models. *In*: **International Conference on Learning Representations (ICLR)**. [*s.n.*], 2023. Disponível em: https://openreview.net/forum?id=WE_vluYUL-X.

ZHENG, L. *et al.* Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. *In*: **Advances in Neural Information Processing Systems**. [*S.l.*: *s.n.*], 2023. v. 36.