Aplicando Ferramentas SAST com IA Gratuitas em uma Ferramenta de Ensino de Gestão de Projetos - Um Benchmarking

Dionas Luan Müller
LESSE/PPGES/UNIPAMPA
Alegrete, RS, Brasil
dionasmuller.aluno@unipampa.edu.br

Matheus Lemes Fialho
LESSE/PPGES/UNIPAMPA
Alegrete, RS, Brasil
matheusfialho.aluno@unipampa.edu.br

Elder de Macedo Rodrigues
LESSE/PPGES/UNIPAMPA
Alegrete, RS, Brasil
elderrodrigues@unipampa.edu.br

Maicon Bernardino
LESSE/PPGES/UNIPAMPA
Alegrete, RS, Brasil
bernardino@acm.org

Resumo

O cenário de segurança de software exige que desenvolvedores integrem ferramentas de Análise Estática de Segurança de Aplicações (SAST) em seus processos de integração contínua (CI/CD). Com a ascensão da Inteligência Artificial (IA) nesse domínio, torna-se crítica a avaliação da acurácia e da usabilidade prática dessas soluções. Este trabalho propõe um benchmarking de validação comparativa para quantificar a eficácia e utilidade de ferramentas SAST freemium com capacidades de IA. A metodologia emprega um *Strong-Sense Benchmark* (SSB) de mais de 150 mil linhas de código em *TypeScript* e o *Framework* Híbrido de Avaliação de SAST (FHA-SAST). O FHA-SAST combina princípios de Verificação e Validação (V&V), métricas de precisão (Verdadeiros/Falsos Positivos) e modelos de aceitação tecnológica (TAM), normalizando todos os resultados a uma escala *Likert* de 1 a 5. Os resultados demonstram um *trade-off* evidente entre ferramentas: o Semgrep alcançou a maior precisão (96,88%), indicando o menor ruído, porém com o maior tempo de execução. Em contraste, o *SonarCloud* obteve a melhor utilidade percebida (5.0), mas com menor precisão objetiva. Concluímos que a adoção ideal de ferramentas SAST/IA exige a análise integrada de desempenho técnico, precisão e fatores logísticos de usabilidade. O FHA-SAST é validado como um método eficaz para guiar a seleção de ferramentas em ambientes de desenvolvimento modernos.

Palavras-chave

Análise Estática de Segurança de Aplicações, Inteligência Artificial, *Benchmarking* de Ferramentas, Segurança de Software, SAST, DevSecOps.

1 Introdução

O cenário contemporâneo de segurança de software apresenta desafios crescentes para organizações em diferentes setores. Estudos empíricos demonstram que vulnerabilidades de software possuem um ciclo de vida médio de aproximadamente 4 anos nos repositórios de código [2], com variações significativas entre projetos (2 anos para o *Chromium* e 7 anos para o *OpenSSL*). Pesquisas recentes indicam que vulnerabilidades são introduzidas através de múltiplas contribuições de código, com 60,93% das vulnerabilidades requerendo mais de um *commit* para serem completamente introduzidas, e janelas de inserção que podem se estender por até 1.520 dias em média [12].

A proliferação de ferramentas de Análise Estática de Segurança de Aplicações (SAST - *Static Application Security Testing*) tornou-se um componente fundamental das estratégias modernas de *DevSecOps*. Zhao *et al.* [33] identificaram cinco dimensões principais do *DevSecOps*, incluindo ferramentas e tecnologias como elementos centrais para a integração efetiva de segurança no ciclo de desenvolvimento. Estudos sistemáticos revelam que a adoção de práticas *DevSecOps* enfrenta 19 desafios distintos, sendo a seleção e integração de ferramentas automatizadas uma das principais barreiras [7].

A integração de Inteligência Artificial (IA) nas ferramentas de segurança representa uma evolução paradigmática no campo. Análises comparativas sistemáticas demonstram que abordagens baseadas em aprendizado de máquina podem complementar ferramentas SAST tradicionais baseadas em regras, oferecendo potencial para redução de falsos positivos e melhoria na detecção de vulnerabilidades [9]. Binbeshr e Imam [6] conduziram uma revisão sistemática da literatura sobre soluções de segurança orientadas por IA em *DevSecOps*, identificando lacunas na validação empírica e na integração prática dessas tecnologias.

Apesar dos avanços tecnológicos, persistem desafios significativos na implementação efetiva dessas soluções. Li *et al.* [13] investigaram perspectivas industriais sobre avaliação de ferramentas SAST, revelando que benchmarks existentes frequentemente falham em atender às necessidades da indústria, impedindo significativamente a adoção de SAST em cenários do mundo real. Estudos empíricos sobre alertas de segurança de ferramentas SAST indicam que desenvolvedores enfrentam dificuldades substanciais devido a altas taxas de falsos positivos, afetando a confiança e adoção dessas ferramentas [4].

A diversidade metodológica entre ferramentas SAST contemporâneas - desde análises tradicionais baseadas em regras até sistemas avançados de aprendizado de máquina - demanda avaliação sistemática para orientar decisões organizacionais. Mapeamentos sistemáticos da literatura identificaram 64 ferramentas SAST distintas cobrindo seis linguagens de programação, avaliadas através de 18 métricas diferentes, evidenciando a complexidade da seleção adequada [28]. Adicionalmente, pesquisas recentes sobre o ciclo de vida de vulnerabilidades revelam que o tempo médio de resposta para correção varia significativamente entre fornecedores de software, com implicações diretas para estratégias de mitigação de riscos [24].

Este artigo apresenta um benchmarking de ferramentas SAST modernas com integração de IA, aplicando um novo framework desenvolvido com base na metodologia de Verificação e Validação de Oberkampf & Trucano [23]— o Framework Híbrido de Avaliação de SAST (FHA-SAST) — para avaliar comparativamente seis soluções: (1) Snyk.io; (2) Aikido Security; (3) Xygeni; (4) Semgrep; (5) SonarCloud; (6) DeepSource. Através da análise do código-fonte da *Silver Bullet*[1] — uma aplicação *TypeScript/React* desenvolvida como ferramenta educacional para o ensino prático de gestão de projetos baseada no *Project Management Body of Knowledge* (PMBOK) [26] — este estudo objetiva fornecer evidências empíricas sobre eficácia, precisão, usabilidade e adequação de cada ferramenta.

Neste contexto, é vital esclarecer a terminologia adotada: o termo *benchmarking* refere-se ao processo metodológico de avaliação comparativa sistemática entre as soluções SAST. Por outro lado, o termo *benchmark* (especificamente, *Strong-Sense Benchmark*) designa a carga de trabalho e o artefato alvo utilizado para basear essa avaliação — neste caso, a plataforma *Silver Bullet*.

Desta forma, para preencher as lacunas identificadas na literatura sobre a avaliação sistemática de soluções SAST baseadas em IA em contextos práticos de desenvolvimento, as principais contribuições deste trabalho para a área de Engenharia de Software e Segurança da Informação são:

- **Adoção de um *Strong-Sense Benchmark* Real:** A utilização de uma aplicação educacional real de alta complexidade (*Silver Bullet*) para a validação empírica, contrapondo-se à dependência tradicional da área em *benchmarks* sintéticos;
- **Framework de Avaliação Híbrido (FHA-SAST):** A proposição e aplicação de um método de avaliação que integra métricas quantitativas de V&V (precisão, revocação, desempenho) com métricas qualitativas de aceitação tecnológica (TAM), permitindo uma visão holística da viabilidade da ferramenta;
- **Evidência Empírica da Troca entre Precisão e Desempenho:** A demonstração quantitativa de que ferramentas de alta precisão (como Semgrep) impõem custos computacionais que podem inviabilizar seu uso síncrono em esteiras de CI/CD ágeis, exigindo novas estratégias de orquestração de segurança;
- **Limitações em Ferramentas *All-in-One*:** A documentação do comportamento anômalo em ferramentas multiúso (como a DeepSource) que, apesar de alta precisão estatística, falham na detecção de vulnerabilidades de código (SAST) em detrimento de análises de estilo e dependências.

2 Referencial Teórico

Esta seção estabelece as bases conceituais sobre análise estática de segurança, os desafios impostos pelas abordagens tradicionais e a emergência da Inteligência Artificial como mecanismo de aprimoramento, além de fundamentar os modelos de avaliação utilizados.

[1]Silver Bullet: https://silverbullet.lesse.com.br/

2.1 Análise Estática de Segurança (SAST) e Desafios de Adoção

A Análise Estática de Segurança de Aplicações (SAST) consiste na inspeção do código-fonte, *bytecode* ou binários de uma aplicação sem a necessidade de executá-la, buscando padrões que correspondam a vulnerabilidades conhecidas [8]. No contexto de metodologias ágeis e *DevSecOps*, o SAST é a principal ferramenta para materializar o conceito de *Shift-Left*, permitindo a identificação precoce de defeitos ainda na fase de codificação.

2.2 Inteligência Artificial Aplicada à Segurança de Código

A integração de técnicas de Inteligência Artificial (IA) e Aprendizado de Máquina (ML) em ferramentas SAST visa mitigar as limitações das abordagens baseadas em regras determinísticas. Modelos baseados em *Deep Learning* e, mais recentemente, Grandes Modelos de Linguagem (LLMs), demonstraram capacidade de aprender representações semânticas do código, permitindo distinguir com maior acurácia entre uma vulnerabilidade real e um falso positivo [3].

Além da detecção, a IA introduz a capacidade de "autorreparação"(*automated program repair*). Ferramentas modernas não apenas apontam a falha, mas sugerem *snippets* de código corrigidos. No entanto, a eficácia dessas correções e a confiabilidade dos modelos em cenários de segurança crítica ainda são objetos de investigação ativa, visto que modelos generativos podem alucinar ou sugerir correções funcionalmente corretas, mas inseguras [25].

2.3 Avaliação de Ferramentas de Software

A avaliação de ferramentas de software transcende a análise puramente técnica. O Modelo de Aceitação de Tecnologia (TAM), proposto por Davis [10], postula que a intenção de uso de um sistema é determinada por duas variáveis principais: a Utilidade Percebida (o quanto a ferramenta melhora o trabalho) e a Facilidade de Uso Percebida (o esforço necessário para usá-la). No contexto de ferramentas de segurança, onde a complexidade é inerente, a aplicação do TAM permite entender por que ferramentas tecnicamente precisas podem ser rejeitadas por times de desenvolvimento se a barreira de entrada for excessiva.

Simultaneamente, a metodologia de Verificação e Validação (V&V) de Oberkampf e Trucano [23], originária da física computacional, fornece o rigor necessário para medir a acurácia dos modelos (neste caso, as ferramentas SAST) frente a um referencial de verdade (o *benchmark*), garantindo que a avaliação técnica seja objetiva e reprodutível.

3 Trabalhos Relacionados

A literatura sobre avaliação de ferramentas de análise estática é extensa, refletindo a importância crítica do SAST no ciclo de desenvolvimento seguro. A evolução desses estudos pode ser categorizada em três vertentes principais: comparações de eficácia, integração de IA e propostas de *benchmarks*.

Em estudos comparativos clássicos, Scandariato *et al.* [27] demonstraram que ferramentas baseadas em análise estática tradicional sofrem com altas taxas de falsos positivos, frequentemente

excedendo 50% dos alertas gerados. Nunes *et al.* [22] corroboraram esses achados ao avaliar ferramentas comerciais e *open-source* contra o *benchmark* sintético OWASP Benchmark, concluindo que nenhuma ferramenta individual era capaz de cobrir todas as categorias de vulnerabilidades (OWASP Top 10) de forma satisfatória.

Com a emergência de técnicas baseadas em dados, a literatura começou a explorar o uso de *Deep Learning* para reduzir o ruído das análises. Li *et al.* [15] propuseram o *VulDeePecker*, um sistema pioneiro de detecção de vulnerabilidades baseado em redes neurais recorrentes (RNNs), demonstrando superioridade sobre ferramentas baseadas em padrões para vulnerabilidades de fluxo de dados. Mais recentemente, Steenhoek *et al.* [29] analisaram o impacto de LLMs na segurança de código, sugerindo que, embora promissores, modelos generativos ainda carecem de compreensão contextual profunda para substituir completamente as *engines* de análise semântica.

Entretanto, uma lacuna persistente identificada por Oberkampf *et al.* [23] e reiterada por Antal *et al.* [5] é a dependência da comunidade científica em *synthetic benchmarks* (códigos artificialmente criados para conter falhas). Embora úteis para testes unitários de ferramentas, esses *benchmarks* falham em reproduzir a complexidade arquitetural, o acoplamento de módulos e as peculiaridades de *frameworks* modernos encontrados na indústria. Este trabalho diferencia-se, primordialmente, pela combinação de duas inovações: a adoção de uma abordagem de Strong-Sense Benchmark — utilizando uma aplicação real de alta complexidade em contraposição aos tradicionais benchmarks sintéticos — e a proposição do Framework Híbrido de Avaliação de SAST (FHA-SAST). Essa integração permite avaliar de forma sistemática não apenas a precisão técnica das ferramentas, mas também sua usabilidade e viabilidade operacional em um cenário real de produção.

4 Metodologia

A abordagem metodológica deste trabalho é estruturada com base nos princípios de Verificação e Validação (V&V), que constituem o principal meio de avaliar a precisão e a confiabilidade de simulações computacionais [23]. O estudo concentra-se especificamente no processo de Validação, cujo objetivo é determinar o quão precisamente um modelo computacional consegue representar fenômenos do mundo real, sob a perspectiva da aplicação final pretendida [23].

Nesse contexto, as ferramentas de Análise Estática de Segurança de Aplicações (SAST) são os "modelos computacionais" em análise. A eficácia de cada ferramenta na identificação de vulnerabilidades corresponde à "representação do mundo real" cuja acurácia foi quantificada. Para garantir uma comparação sistemática e objetiva, o procedimento emprega *benchmarks* rigorosamente definidos, alinhando-se às diretrizes para a construção de avaliações de V&V de alta qualidade.

4.1 Objetivo e Questões de Pesquisa

O objetivo principal é realizar uma análise de validação comparativa para quantificar a acurácia e a utilidade de diferentes ferramentas SAST em nuvem, de acesso gratuito ou com planos *freemium*, que possuam algum nível de integração com IA. Para guiar a investigação, foram formuladas as seguintes Questões de Pesquisa (QP) (Figura 1).

QP1. Quais ferramentas identificam a maior quantidade e variedade de vulnerabilidades?
QP2. Quais ferramentas apresentam maior precisão, avaliada pela sua capacidade de minimizar a ocorrência de falsos positivos?
QP3. Quais ferramentas apresentam os resultados de forma mais clara e fornecem o melhor suporte para a compreensão e remediação das falhas identificadas?
QP4. Quais ferramentas oferecem a melhor usabilidade em seu ciclo de vida completo (configuração, execução e análise dos resultados)?

Figura 1: Questões de Pesquisa (QP).

4.2 Seleção e Caracterização da Aplicação-Base

Para garantir a reprodutibilidade e a validade das comparações, o objeto de análise deste estudo — o código-fonte do projeto *Silver Bullet* — é tratado como um *Strong-Sense Benchmark* (SSB) [23]. Suas características são definidas a seguir:

- **Propósito e Escopo do Benchmark:** O benchmark tem como propósito avaliar o desempenho de ferramentas SAST em um projeto de software moderno, escrito predominantemente em **TypeScript** [19], utilizando os *frameworks* **React** [18] e **Node.js** [21] (com NodeTS), com arquitetura baseada em *Clean Architecture* [17]. Os resultados são diretamente relevantes para a análise de segurança de aplicações web de complexidade similar.
- **Descrição Precisa do Benchmark:** Foi utilizada a versão estável mais recente do repositório oficial da *Silver Bullet*. O projeto possui mais de **150 mil Linhas de Código (LOC)**. A análise abrangeu todos os módulos e componentes do núcleo do projeto, excluindo arquivos de documentação.
- **Quantificação de Incertezas:** A "incerteza" do benchmark refere-se a características do código que representam desafios conhecidos para a análise estática automatizada. Neste estudo, foram pré-mapeadas as seguintes fontes de incerteza: a) áreas de alta complexidade, que podem dificultar a análise de fluxo de controle; e b) interdependências não triviais entre módulos da aplicação, que desafiam a capacidade das ferramentas de realizar análises de fluxo de dados intercomponentes.
- **Documentação:** Como o código-fonte da *Silver Bullet* é privado, o foco da documentação foi garantir a **reprodutibilidade interna** e a **auditabilidade** do estudo. Para isso, o código-fonte foi isolado em um repositório Git dedicado. A fim de assegurar a auditabilidade, os seguintes artefatos foram arquivados e preservados: o **hash de commit** (1336fed492547acfab1767bbaa7e6f3730e6e66c) que define a versão do benchmark, os arquivos de *lock* de dependências (*e.g.*, `package-lock.json`) e os relatórios brutos de cada ferramenta. Adicionalmente, os resultados e os artefatos gerados pelas análises foram armazenados em um segundo repositório, separado do código-fonte, para evitar qualquer interferência com ferramentas que necessitam de acesso direto ao repositório do benchmark.

4.3 Procedimento Experimental de Validação

O procedimento experimental foi elaborado para garantir comparações justas e minimizar vieses.

4.3.1 Ferramentas Candidatas. As seguintes ferramentas foram selecionadas com base nos critérios de: (i) disponibilidade gratuita ou *freemium*; (ii) realizar análise SAST com capacidades de IA integradas; (iii) modelo SaaS que dispense configuração de infraestrutura local; (iv) possibilidade de execução em repositórios privados.

- Snyk.io
- Aikido Security
- Xygeni
- Semgrep
- SonarCloud
- DeepSource

4.3.2 Configuração e Execução. Uma distinção rigorosa entre validação e calibração é fundamental. A calibração envolve o ajuste de parâmetros para melhorar a concordância com dados conhecidos, enquanto a validação busca avaliar a capacidade preditiva do modelo de forma independente [23]. Para aderir a esse princípio, todas as ferramentas foram executadas com suas configurações padrão recomendadas. Nenhuma regra foi ajustada ou desabilitada após uma análise inicial com o objetivo de "melhorar" os resultados, garantindo uma avaliação imparcial da eficácia padrão de cada solução.

4.3.3 Coleta de Dados. O processo de coleta seguiu um protocolo estrito:

- Os relatórios brutos de cada ferramenta foram exportados em seus formatos originais (JSON, CSV ou PDF) e armazenados.
- O tempo de execução de cada análise, desde o início do processo até a geração do relatório final, foi cronometrado.
- Durante o uso da ferramenta, foram registradas observações qualitativas sobre a experiência de utilização e imediatamente após foi respondido um questionário com as questões definidas nas seção a seguir (métricas QP3.M1 e QP4.M1).

4.4 O *Framework* Híbrido de Avaliação de SAST (FHA-SAST)

Para responder às QPs de forma sistemática, este estudo propõe o *Framework* Híbrido de Avaliação de SAST (FHA-SAST). Este método combina métricas de desempenho técnico (eficácia, precisão) com modelos consolidados de aceitação de tecnologia (utilidade e usabilidade), reconhecendo que a adoção prática de uma ferramenta é tão crucial quanto sua precisão. O *framework* é definido pelo mapeamento direto das Questões de Pesquisa (QPs) para um conjunto de Métricas de Validação (M) quantitativas e qualitativas.

4.4.1 Definição das Métricas de Validação. A comparação dos resultados foi realizada por meio de métricas específicas para cada questão de pesquisa:

- **Métricas para QP1 (Capacidade de Detecção):**

QP1.M1. Contagem de Verdadeiros Positivos (VP).
QP1.M2. Contagem de Falsos Positivos (FP).
QP1.M3. Número total de vulnerabilidades reportadas.
QP1.M4. Contagem de achados por nível de severidade.

- **Métrica para QP2 (Precisão):**

QP2.M1. Precisão (Accuracy), calculada por: $TP/(TP + FP)$.

- **Métricas para QP3 (Utilidade e Clareza) - Qualitativas (Likert 1-5):**

QP3.M1. Eficácia na detecção de vulnerabilidades (percepção subjetiva).
QP3.M2. Qualidade e contextualização dos resultados.
QP3.M3. Valor agregado para o processo de desenvolvimento.
QP3.M4. Facilidade de aplicação das correções sugeridas pela ferramenta.
QP3.M5. Qualidade da documentação e suporte.

- **Métricas para QP4 (Usabilidade e Desempenho) - Mistas:**

QP4.M1. Tempo de execução da análise (Quantitativo, medido em hh:mm:ss).
QP4.M2. Simplicidade de configuração e integração (Likert 1-5).
QP4.M3. Clareza da interface e navegação (Likert 1-5).
QP4.M4. Confiabilidade dos resultados (Likert 1-5).
QP4.M5. Reprodutibilidade das análises (Likert 1-5).
QP4.M6. Aderência às capacidades declaradas (Likert 1-5).

4.4.2 Fundamentação Teórica das Métricas. A estrutura de avaliação adotada combina métricas quantitativas objetivas com avaliações qualitativas baseadas em modelos consolidados da literatura. Para QP1 e QP2, foram definidas métricas puramente quantitativas, enquanto QP3 e QP4 integram o *Technology Acceptance Model* (TAM) [10] com o *framework* de avaliação de Li *et al.* [14].

Technology Acceptance Model (TAM). O TAM [10] fundamenta a avaliação de aceitação tecnológica através de dois construtos principais: *Perceived Usefulness* (utilidade percebida) — grau em que o usuário acredita que a tecnologia melhora seu desempenho — e *Perceived Ease of Use* (facilidade de uso percebida) — grau em que o usuário considera o uso da tecnologia livre de esforço. Esses construtos foram adaptados ao contexto de ferramentas SAST, onde utilidade relaciona-se à eficácia na detecção e ao valor para o processo de desenvolvimento, enquanto facilidade de uso abrange configuração, integração e navegação.

Framework de Li et al.: Li *et al.* [14] propuseram um *framework* para avaliação de ferramentas SAST estruturado em três dimensões: *Effectiveness* (eficácia, medida por precisão e recall), *Performance* (tempo de execução) e *Consistency* (concordância entre capacidades declaradas e resultados reais). Os autores utilizaram o *Common Weakness Enumeration* (CWE) como referência para mapear regras de detecção e vulnerabilidades, permitindo comparação sistemática entre ferramentas. Especificamente, mapearam 1.801 regras de sete ferramentas SAST e vulnerabilidades de dois *benchmarks* para CWE Classes do CWE-1000, unificando o nível hierárquico de análise.

Integração e Adaptação: A estrutura proposta integra os construtos do TAM com as dimensões de Li *et al.*, resultando em quatro categorias: **Utilidade Percebida** (eficácia, qualidade dos resultados, valor agregado), **Facilidade de Uso Percebida** (configuração, interface, documentação), **Desempenho Técnico** (tempo de execução e facilidade de aplicação de correções — esta última adaptada ao contexto moderno de ferramentas com recursos de IA para geração automática de PRs), e **Consistência** (confiabilidade, reprodutibilidade, aderência às capacidades declaradas). Todas as métricas

qualitativas utilizam escala Likert de 1 a 5, conforme práticas consolidadas na literatura [10].

4.4.3 Normalização de Métricas Quantitativas para Escala Likert. A avaliação integrada de métricas quantitativas e qualitativas apresenta o desafio de comparar diferentes tipos de dados em uma escala unificada. Para contornar essa limitação, adotou-se o procedimento de normalização de métricas quantitativas para a escala Likert de 1 a 5, conforme recomendações de Stevens [30] para transformação de dados ordinais.

Normalização do Tempo de Execução. O tempo de execução foi normalizado através do método de comparação com a média de referência, amplamente utilizado em benchmarking de performance [11]. Calculou-se a média aritmética dos tempos de todas as ferramentas:

$$\bar{t} = \frac{1}{n} \sum_{i=1}^{n} t_i$$

onde t_i é o tempo de execução da ferramenta i e n é o número total de ferramentas. Para cada ferramenta, calculou-se o desvio percentual em relação à média:

$$\mathrm{d}(\%) = \frac{t_i - \bar{t}}{\bar{t}} \times 100$$

Posteriormente, este desvio (d) percentual foi mapeado para a escala Likert 1-5, onde valores de d negativos indicam que a ferramenta i é mais rápida que a média, e positivos, mais lenta. Assim:

- **5 pontos:** $d < -50\%$ (muito mais rápida que a média)
- **4 pontos:** d entre -50% e -20%
- **3 pontos:** d entre -19% e 20% (próxima da média)
- **2 pontos:** d entre 21% e 50%
- **1 ponto:** $d > 50\%$ (muito mais lenta que a média)

Esta normalização é justificada pela abordagem de "benchmarking com baseline comparativo", onde o desempenho relativo é mais informativo que valores absolutos [11].

Normalização da Precisão. A precisão foi normalizada através de mapeamento linear direto dos valores de $TP/(TP+FP)$ para a escala Likert, seguindo a recomendação de Agresti [1] para transformação de proporções em escalas ordinais:

- **1 ponto:** Precisão < 20% (confiança muito baixa nas detecções)
- **2 pontos:** Precisão entre 20% e 40%
- **3 pontos:** Precisão entre 40% e 60%
- **4 pontos:** Precisão entre 60% e 80%
- **5 pontos:** Precisão > 80% (alta confiança)

Este esquema de mapeamento reflete a prática comum em avaliações de sistemas de recuperação de informação [16], onde precisão acima de 80% é considerada excelente, entre 60-80% é boa, entre 40-60% é aceitável, e abaixo de 40% é considerada problemática para uso prático [14].

Integração e Interpretação. A normalização de métricas quantitativas permite comparação unificada com avaliações qualitativas (Likert direto) sem perder a natureza e significado dos dados originais. Este procedimento segue a recomendação de Sullivan [31] de que escalas Likert com 5 ou mais pontos podem incorporar dados quantitativos normalizados através de mapeamento linear. O resultado final é uma matriz comparativa onde todas as dimensões (desempenho, usabilidade, consistência e utilidade) estão na mesma escala ordinal 1-5, facilitando análise agregada e ranking das ferramentas.

Cabe ressaltar que, para fins de consolidação, este estudo adota uma premissa *unweighted* (pesos iguais) para todas as métricas normalizadas. Trabalhos futuros poderão empregar métodos de Análise de Decisão Multicritério, atribuindo pesos diferentes conforme a prioridade organizacional (e.g., priorizar velocidade de execução versus precisão).

4.4.4 Análise e Classificação de Achados. Para a análise manual dos alertas gerados pelas ferramentas e para garantir a consistência da avaliação de precisão (QP2), os seguintes critérios de classificação foram definidos:

- **Verdadeiro Positivo (VP):** Um alerta foi classificado como VP quando, após inspeção manual do código-fonte, confirmou-se que o alerta indicava:
  - Uma vulnerabilidade real e explorável no código da aplicação;
  - Uma má prática de segurança significativa no contexto da aplicação (e.g., código que, embora não diretamente explorável, viola princípios de defesa em profundidade); ou
  - Uma vulnerabilidade conhecida em uma dependência (biblioteca) do projeto, independentemente de a funcionalidade vulnerável estar em uso ativo pela aplicação.
- **Falso Positivo (FP):** Um alerta foi classificado como FP quando a inspeção manual revelou que o código apontado:
  - Não possuía uma vulnerabilidade (o alerta era factualmente incorreto);
  - Era irrelevante para o contexto de segurança da aplicação; ou
  - A ferramenta interpretou o código de forma incorreta (*e.g.*, apontando código de arquivos de teste como uma vulnerabilidade de produção).
- **Duplicata:** Um alerta foi classificado como duplicata se a ferramenta gerou múltiplos relatórios para a mesma instância de vulnerabilidade subjacente. Achados distintos, mesmo que em locais próximos (como linhas diferentes no mesmo arquivo, apontando instâncias diferentes do mesmo tipo de falha), não foram considerados duplicatas. Os alertas classificados como duplicatas foram contados apenas uma vez na apuração de VPs ou FPs.

  Para mitigar o viés subjetivo inerente à avaliação qualitativa, o processo de inspeção manual e classificação dos alertas foi conduzido de forma independente por dois pesquisadores. O perfil dos avaliadores inclui estudantes de pós-graduação stricto sensu (Mestrado) com experiência ativa na indústria como desenvolvedores de software *full-stack*, garantindo o alinhamento necessário entre o rigor acadêmico e a vivência prática em desenvolvimento web e estruturação de dados.

Em relação à classificação dos achados, o estudo adotou a abordagem metodológica de consenso por pares (*peer consensus*). Devido à natureza interpretativa do contexto arquitetural da aplicação frente às vulnerabilidades apontadas, eventuais divergências na classificação inicial (VP, FP ou Duplicata) foram tratadas qualitativamente através de deliberações técnicas aprofundadas. Durante essas sessões de conciliação, cada caso conflitante foi analisado conjuntamente à luz das definições da *Common Weakness Enumeration* (CWE) e da viabilidade prática de exploração da falha, até que um acordo unânime fosse estabelecido com base em argumentação técnica e arquitetural.

4.5 Limitações do Estudo

Este benchmarking possui as seguintes limitações inerentes: (i) os resultados são específicos para projetos com arquitetura similar à *Silver Bullet*; (ii) a análise manual de falsos positivos está sujeita à interpretação dos pesquisadores; (iii) vulnerabilidades do tipo *runtime* não são contempladas por ferramentas SAST; (iv) a avaliação de usabilidade reflete a perspectiva de desenvolvedores com experiência específica em *TypeScript/React*.

Adicionalmente, por adotar um *Strong-Sense Benchmark* baseado em um projeto de software real, este estudo não dispõe de um *ground truth* absoluto contendo todas as vulnerabilidades pré-existentes no código. Consequentemente, não é possível mensurar com exatidão os Falsos Negativos, *i.e.* as falhas que as ferramentas falharam em detectar, impossibilitando o cálculo preciso da métrica de Revocação (*Recall*). A avaliação quantitativa restringe-se, portanto, à classificação do volume de alertas que as ferramentas efetivamente reportaram, categorizando-os em Verdadeiros e Falsos Positivos.

Apesar desta limitação impossibilitar o cálculo exato da métrica de Revocação (Recall), reconhecemos que a criação de uma baseline a partir da união de todos os Verdadeiros Positivos seria uma abordagem promissora. Contudo, devido às restrições dos planos freemium e trial utilizados neste benchmarking — que frequentemente bloqueiam a exportação detalhada de relatórios e revogam o acesso aos dados após curtos períodos (ex: 7 a 14 dias) —, a validação cruzada granular entre as ferramentas tornou-se inviável pós-execução. A aplicação dessa estratégia analítica, com o uso de licenças adequadas para retenção de dados, está planejada como escopo prioritário para desdobramentos futuros.

5 Análise dos Resultados

Esta seção apresenta os resultados comparativos das ferramentas SAST analisadas. Os dados foram coletados seguindo o protocolo descrito na Seção 4 e são apresentados de forma a responder cada Questão de Pesquisa (QP).

5.1 Resultados Quantitativos: Detecção e Variedade (QP1)

A primeira questão de pesquisa (QP1) buscou avaliar a capacidade de detecção de cada ferramenta. Os resultados brutos da análise, incluindo a contagem total de vulnerabilidades, Verdadeiros Positivos (VP), Falsos Positivos (FP) e a distribuição por severidade, estão consolidados na Tabela 1.

Tabela 1: Resultados quantitativos da detecção de vulnerabilidades por ferramenta.

Ferramentas	VP	FP	Total	Muito Alta	Alta	Média	Baixa
Snyk	6	15	21	0	1	20	0
Aikido Security	9	61	70	0	37	27	6
Xygeni	39	20	59	7	17	0	35
Semgrep	31	1	32	1	7	18	6
SonarCloud	18	34	52	0	24	21	7
DeepSource	28	0	28	1	5	16	6

Legenda: VP = Verdadeiros Positivos, FP = Falsos Positivos.

Os dados brutos mostram uma clara distinção entre as ferramentas em termos de volume. O *Aikido Security* reportou o maior volume total de alertas (70), enquanto o *Xygeni* e o *Semgrep* lideraram em termos de Verdadeiros Positivos (39 e 31, respectivamente), indicando maior eficácia em detecção de falhas reais.

5.2 Resultados Quantitativos: Precisão (QP2)

A segunda questão de pesquisa (QP2) avaliou a precisão das ferramentas, medida pela capacidade de minimizar falsos positivos. A precisão foi calculada usando a fórmula padrão Precisão = $VP/(VP+FP)$, com base nos dados classificados manualmente. Os resultados estão na Tabela 2.

Tabela 2: Comparativo de Precisão das ferramentas (QP2).

Ferramentas	Verdadeiros Positivos	Falsos Positivos	Precisão Calculada
Snyk	6	15	28,60%
Aikido Security	9	61	12,86%
Xygeni	39	20	66,10%
Semgrep	31	1	96,88%
SonarCloud	18	34	34,62%
DeepSource*	28	0	100%

Observação: *Vulnerabilidades detectadas referem-se exclusivamente a dependências (SCA)

A precisão (QP2) é o aspecto mais contrastante do estudo. O *Semgrep* demonstrou uma precisão superior de 96,88%, destacando-se como o mais confiável no contexto analisado. Em contraste, *Aikido Security* e *Snyk* apresentaram taxas de precisão abaixo de 30%, resultando em alto ruído para o desenvolvedor. Esses resultados de baixa precisão, em muito se devem à muitas das ferramenta considerarem alguns dos testes que possuíam senhas explícitas no código como falhas de segurança, o que foi considerado como falso positivo.

5.3 Resultados Qualitativos: Clareza e Usabilidade (QP3 e QP4)

As questões QP3 (Clareza e suporte à remediação) e QP4 (Usabilidade) foram avaliadas por meio de métricas qualitativas (baseadas no TAM e no *framework* de Li *et al.*) e pelo tempo de execução das análises.

5.3.1 Desempenho da Análise (QP4). Um componente da usabilidade (QP4) é o desempenho técnico, medido pelo tempo de execução

Tabela 3: Resultados da avaliação qualitativa (QP3 e QP4).

Métricas	Snyk	Aikido	Xygeni	Semgrep	SonarCloud	DeepSource
Utilidade Percebida (Média)	**3,66**	**4**	**3,33**	**4,33**	**5**	**4,66**
Eficácia na detecção	3	2	4	5	5	4
Qualidade dos resultados	4	5	3	4	5	5
Valor Agregado ao desenvolvimento	4	5	3	4	5	5
Facilidade de Uso (Média)	**4,66**	**4,66**	**3**	**3,66**	**3,33**	**3,33**
Simplicidade de configuração	5	5	5	3	2	3
Clareza da Interface	4	4	2	3	3	2
Qualidade da Documentação e Suporte	5	5	5	5	5	5
Desempenho Técnico (Média)	**3,66**	**3,66**	**3**	**3**	**3,66**	**3,66**
Facilidade de aplicação das correções	4	5	2	3	4	5
Precisão (falsos positivos/negativos)	2	1	4	5	2	5
Tempo de execução da análise	5	5	3	1	5	1
Consistência (Média)	**4,33**	**4,33**	**4**	**4,33**	**4**	**3,33**
Confiabilidade dos resultados	3	3	4	5	4	3
Reprodutibilidade das análises	5	5	5	5	4	3
Aderência às capacidades declaradas	5	5	3	3	4	4
Média das Métricas	**4,08**	**4,17**	**3,58**	**3,83**	**4,00**	**3,75**

Tabela 4: Observações qualitativas sobre limitações e diferenciais.

Aspectos	Snyk	Aikido	Xygeni	Semgrep	SonarCloud	DeepSource
Exportação de relatórios	–	–	–	–	–	X
Identifica falsos positivos nativamente		X			X	
Sugestão de correção com IA	–	X	–	–	–	X
Apresentou resultados duplicados			X	X	X	
Separa erros em código vs bibliotecas				X		X
Ignora corretamente arquivos de teste				X		X
Permite análise de dependências	X	X	X	X		X
Possui análise de uso real de biblioteca		X		X		X
Encontrou vulnerabilidades na análise de código	X	X	X	X	X	
Plano utilizado	**Free**	**Free**	**7d-Trial**	**7d-Trial**	**14d-Trial**	**Free**

Legenda: X: Característica presente (Disponível no Plano Testado). —: Característica disponível apenas para planos pagos. **(Célula em branco)**: Característica não presente ou não aplicável ao escopo da ferramenta.

Tabela 5: Tempo de execução das análises (QP4).

Ferramentas	Tempo de Execução
Snyk	00:01:48
Aikido Security	00:01:27
Xygeni	00:08:36
Semgrep	00:26:55
SonarCloud	00:01:34
DeepSource	00:18:28

da análise em nuvem. Os tempos foram cronometrados desde o início da análise até a disponibilização do relatório e estão na Tabela 5.

O tempo de execução é um fator crítico para a QP4. *Aikido Security* e *SonarCloud* foram os mais ágeis, com tempos de análise abaixo de 1 minuto e 40 segundos. O *Semgrep*, apesar da alta precisão (QP2), exigiu o maior tempo de processamento, o que pode ser um obstáculo em ambientes de integração contínua (CI/CD) com janelas curtas.

5.3.2 Utilidade e Facilidade de Uso Percebidas (QP3 e QP4). A avaliação de Utilidade Percebida (QP3) e Facilidade de Uso Percebida (QP4) foi realizada através dos questionários (Likert 1-5) definidos no protocolo. Os resultados consolidados dessa avaliação estão apresentados na Tabela 3.

A avaliação qualitativa revelou um notável alinhamento entre a percepção de precisão e os dados quantitativos (QP2), onde as ferramentas de maior precisão calculada (Semgrep) também obtiveram as maiores notas de confiabilidade. A análise demonstrou um *trade-off* clássico entre potência e usabilidade: o *SonarCloud* obteve a pontuação máxima em Utilidade Percebida (5,0), refletindo a qualidade de seus resultados, mas ficou com a pior nota em simplicidade de configuração (2,0). Em contraste, o *Aikido Security* e o *Snyk* obtiveram as melhores notas em Facilidade de Uso (4,66), porém, sua baixa precisão (Tabela 2) impactou negativamente a percepção de eficácia.

5.3.3 Observações Qualitativas da Execução. Para complementar as métricas de usabilidade e utilidade (QP3 e QP4), a Tabela 4 mapeia as características funcionais e as restrições de plano observadas em cada ferramenta. Esta análise é crítica para desenvolvedores que buscam integrar soluções SAST em ambientes de código aberto ou com recursos limitados, uma vez que funcionalidades essenciais (como a exportação de relatórios) são frequentemente condicionadas ao pagamento.

6 Discussão dos Resultados

Esta seção interpreta os resultados obtidos à luz das quatro Questões de Pesquisa (QP) formuladas, contextualizando o desempenho das ferramentas frente aos desafios reais de desenvolvimento seguro.

Cabe destacar que a maioria das soluções avaliadas opera como um modelo de caixa preta (*black-box*) em relação aos seus motores

de análise. Consequentemente, não é possível segregar quantitativamente quais vulnerabilidades foram detectadas por heurísticas tradicionais baseadas em regras e quais foram identificadas exclusivamente pelos módulos de IA subjacentes. A análise reflete, portanto, a eficácia composta (IA e motores tradicionais) das ferramentas oferecidas ao usuário final.

6.1 Discussão da QP1: Capacidade e Variedade de Detecção

A análise da QP1 revela que *volume* de alertas não se traduz necessariamente em *segurança*. O *Aikido Security* reportou a maior quantidade absoluta de vulnerabilidades (70), mas a taxa de ruído (falsos positivos) diluiu a eficácia dessa detecção. Em contrapartida, *Xygeni* e *Semgrep* demonstraram maior robustez na identificação de falhas de código complexas (Verdadeiros Positivos), validando a superioridade de seus motores de análise semântica.

Um destaque negativo crítico nesta QP foi a ferramenta *DeepSource*. Embora o estudo focasse em SAST (análise de código), a ferramenta falhou em identificar qualquer vulnerabilidade na lógica da aplicação, restringindo seus achados exclusivamente a problemas de dependências (SCA). Isso indica uma limitação severa de abrangência (*recall*) para o contexto de segurança de código, comportando-se, na prática, como uma ferramenta de SCA e *Linter*, e não como uma solução SAST completa para o *benchmark* proposto.

6.2 Discussão da QP2: Precisão e Confiabilidade

A QP2 evidenciou a disparidade na inteligência contextual das ferramentas. O *Semgrep* atingiu o patamar de excelência (96,88% de precisão) principalmente por sua capacidade de distinguir contextos: ele ignorou corretamente credenciais *hardcoded* em arquivos de teste, enquanto ferramentas como *Snyk* e *Aikido* as flagraram como críticas, derrubando suas precisões para abaixo de 30%.

O caso da *DeepSource* requer uma interpretação cautelosa nesta métrica. A ferramenta obteve, estatisticamente, 100% de precisão (0 falsos positivos). No entanto, este resultado é consequência de sua "cegueira seletiva" discutida na QP1: ao não reportar falhas de código, ela eliminou a possibilidade de erro, mas também a de acerto nessa categoria. Este fenômeno reforça que a métrica de precisão, quando isolada da capacidade de detecção, pode mascarar a ineficácia de uma ferramenta. Para fins práticos de SAST, a alta precisão do *Semgrep* é valiosa, enquanto a da *DeepSource* é um artefato estatístico decorrente de sua baixa sensibilidade.

6.3 Discussão da QP3: Clareza, Utilidade e Suporte

Na avaliação qualitativa (QP3), observou-se que a percepção de utilidade está atrelada mais à qualidade da informação do que à facilidade de uso. O *SonarCloud* liderou este quesito (nota 5,0 em Utilidade Percebida), demonstrando que desenvolvedores valorizam ferramentas que não apenas apontam o erro, mas contextualizam o risco e educam sobre a correção.

As funcionalidades de IA Generativa para "autofix" (presentes no *Aikido* e *DeepSource*) foram bem avaliadas qualitativamente, representando uma tendência promissora para reduzir o tempo de remediação. No entanto, a utilidade real dessas sugestões depende da precisão da detecção base; sugerir correções para falsos positivos (como ocorreu frequentemente com o *Aikido*) pode gerar trabalho desnecessário, mitigando o ganho de utilidade.

6.4 Discussão da QP4: Usabilidade e Ciclo de Vida

A QP4 revelou um *trade-off* claro entre profundidade de análise e tempo de execução. O *Semgrep*, embora o mais preciso, foi significativamente mais lento (aproximadamente 27 minutos) do que a média das demais ferramentas (< 2 minutos). Isso impõe restrições ao seu uso em esteiras de CI/CD síncronas (bloqueio de PRs), sugerindo seu posicionamento ideal em análises assíncronas ou *nightly builds*.

Por outro lado, ferramentas como *Snyk* e *Aikido* ofereceram uma experiência *plug-and-play* superior e tempos de resposta rápidos, favorecendo a adoção inicial e o feedback imediato, ainda que com menor confiabilidade. O *SonarCloud* apresentou o perfil de ferramenta "Enterprise": barreira de entrada mais alta na configuração (nota 2,0), mas excelente usabilidade e gestão a longo prazo após a integração, confirmando que a complexidade inicial é tolerada em troca de robustez operacional.

A *DeepSource*, apesar de sua interface moderna e recursos de IA, apresentou uma relação custo-benefício desfavorável neste estudo: exigiu um tempo de análise alto (18 minutos) e configuração complexa, sem entregar a contrapartida esperada na detecção de falhas de segurança de código.

Por fim, um aspecto crítico observado quanto ao ciclo de vida e usabilidade (QP4) em ferramentas comerciais com modelos *freemium* ou *trials* (como Xygeni e Semgrep) é a volatilidade e o *lock-in* dos dados. O acesso ao histórico detalhado de detecções e categorizações específicas de vulnerabilidades (CWE) frequentemente é revogado ou expurgado pelos provedores após o término do período de avaliação. Para projetos de código aberto ou pesquisas acadêmicas que demandam auditorias longitudinais, essa restrição de retenção de dados impõe uma barreira significativa, forçando as equipes a dependerem de exportações manuais imediatas — funcionalidade que, como mapeado na Tabela 5, costuma ser restrita aos planos pagos.

7 Ameaças à Validade

Seguindo as diretrizes de Wohlin *et al.* [32], discutimos as ameaças à validade deste estudo e as estratégias de mitigação adotadas.

7.1 Validade Interna

A principal ameaça à validade interna reside na classificação manual dos alertas em Verdadeiros ou Falsos Positivos, cuja interpretação pode ser intrinsecamente subjetiva. Esse risco é amplificado pelo viés metodológico inerente ao conhecimento prévio dos avaliadores sobre a arquitetura e o código-fonte da aplicação utilizada como *benchmark*. Para mitigar ambas as ameaças, a classificação e a validação cruzada dos achados seguiram estritamente as diretrizes objetivas da *Common Weakness Enumeration* (CWE). Em casos de dúvida, buscou-se a prova de conceito (PoC) da explorabilidade da falha no contexto da aplicação, reduzindo a margem para interpretações enviesadas e garantindo uma análise puramente técnica. Além disso, a configuração das ferramentas limitou-se aos padrões de

fábrica (*defaults*), o que pode não refletir o desempenho otimizado de cada solução, mas assegura a imparcialidade da comparação *"out-of-the-box"*.

7.2 Validade Externa

Os resultados obtidos restringem-se ao ecossistema *TypeScript/React* e à arquitetura específica do projeto alvo do estudo. Não é possível generalizar as conclusões para linguagens com tipagem estática forte (como Java ou C++) ou arquiteturas monolíticas tradicionais. A escolha de um único projeto, embora aprofundada, limita a variabilidade da amostra.

7.3 Validade de Construto

As métricas do *Technology Acceptance Model* (TAM) baseiam-se na percepção subjetiva dos pesquisadores enquanto usuários das ferramentas. Embora guiadas por um questionário estruturado, essas avaliações podem ser influenciadas por familiaridade prévia com certas interfaces ou vieses cognitivos. Buscou-se mitigar isso através da triangulação com métricas objetivas (tempo de execução e precisão de verdadeiros positivos).

8 Conclusão

Este trabalho apresentou um benchmarking sistemático de seis ferramentas SAST modernas com capacidades de IA, avaliando-as sob as perspectivas de eficácia, precisão, utilidade e usabilidade através do framework híbrido FHA-SAST. A utilização do projeto de estudo omitido como *Strong-Sense Benchmark* permitiu isolar o desempenho das ferramentas em um cenário de desenvolvimento moderno e complexo.

Em resposta às Questões de Pesquisa, o estudo conclui que:

- **Sobre a Capacidade de Detecção (QP1):** Não existe uma ferramenta dominante. Enquanto o *Xygeni* e o *Semgrep* demonstraram superioridade na identificação de falhas complexas de código, ferramentas como a *DeepSource* falharam em entregar capacidades de SAST para a lógica da aplicação, limitando-se à análise de dependências e estilo;
- **Sobre a Precisão (QP2):** O *Semgrep* estabeleceu-se como o estado da arte em precisão (96,88%), validando a importância de análises sensíveis ao contexto (como a exclusão automática de arquivos de teste). Em contraste, ferramentas focadas em volume (*Aikido*, *Snyk*) apresentaram taxas de ruído que podem inviabilizar sua adoção sem configuração e calibração extensivas;
- **Sobre a Utilidade e Clareza (QP3):** O *SonarCloud* provou que a contextualização dos resultados é tão importante quanto a detecção, obtendo a máxima utilidade percebida. A clareza na explicação do risco mitiga a complexidade da remediação;
- **Sobre a Usabilidade e Ciclo de Vida (QP4):** Identificou-se um *trade-off* crítico entre tempo e profundidade. Ferramentas de alta precisão (*Semgrep*) exigiram tempos de execução (aprox. 27 min) incompatíveis com fluxos de *Pull Request* bloqueantes, enquanto soluções rápidas (*Snyk*, *Aikido*) sacrificaram a precisão pela agilidade.

Como implicação prática para a indústria, este estudo sugere que a adoção sustentável de SAST em esteiras *DevSecOps* exige uma orquestração híbrida e consciente. Recomenda-se o uso de ferramentas de varredura rápida e baixo atrito (como *SonarCloud* ou *Snyk*) em verificações síncronas por *commit*, fornecendo *feedback* imediato sem interromper o fluxo ágil. Em contrapartida, soluções de alta precisão semântica e maior custo computacional (como *Semgrep* ou *Xygeni*) devem ser alocadas em execuções assíncronas ou processos noturnos (*nightly builds*). Adicionalmente, as organizações devem planejar estrategicamente a gestão de vulnerabilidades, considerando o risco de *lock-in* de dados em modelos *freemium* e o impacto da fadiga de alertas causada por altas taxas de falsos positivos — fatores logísticos e cognitivos que, conforme fundamentado pelo FHA-SAST, afetam diretamente a aceitação dessas tecnologias pelas equipes de desenvolvimento. Por fim, recomenda-se extrema cautela na adoção de ferramentas '*all-in-one*' sem a validação rigorosa de suas capacidades específicas de SAST, como evidenciado pelo caso da *DeepSource*. Embora essas soluções agreguem valor significativo ao projeto por meio de suas outras funcionalidades, elas podem não ser a escolha mais adequada para contextos em que a segurança é um requisito prioritário.

Para trabalhos futuros, propõe-se a expansão deste *benchmarking* para outras linguagens de programação e a avaliação específica da qualidade do código gerado pelas funcionalidades de "autofix" com IA, verificando se as correções sugeridas não introduzem novas vulnerabilidades ou débitos técnicos.

Disponibilidade de Artefatos

Estamos empenhados em promover a transparência e a reprodutibilidade na investigação. Seguindo esse compromisso, fornecemos a base de dados completa, relatórios brutos de dados de execução que apoiam as descobertas do nosso estudo. Estes artefatos estão licenciados sob a Creative Commons Attribution 4.0 International (CC BY 4.0) e encontram-se disponíveis abertamente no Zenodo em https://doi.org/10.5281/zenodo.21866721 [20].

Agradecimentos

Os autores agradecem à Universidade Federal do Pampa (Unipampa) pelo apoio durante o desenvolvimento deste trabalho. Dionas Müller agradece à Pró-Reitoria de Pesquisa e Pós-graduação (PROPPI) da Unipampa pelo apoio do Edital Interno PROPPI nº 11/2025 - Programa de Auxílio da Pós-Graduação (PAPG). Maicon Bernardino é financiado pelo Conselho Nacional de Desenvolvimento Científico e Tecnológico (Bolsa CNPq #307969/2026-6).

Referências

[1] Alan Agresti. 2010. *Analysis of ordinal categorical data*. Vol. 656. John Wiley & Sons.

[2] Nikolaos Alexopoulos, Manuel Brack, Jan Philipp Wagner, Tim Grube, and Max Mühlhäuser. 2022. How Long Do Vulnerabilities Live in the Code? A Large-Scale Empirical Measurement Study on FOSS Vulnerability Lifetimes. In *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, Boston, MA, 359–376. https://www.usenix.org/conference/usenixsecurity22/presentation/alexopoulos

[3] Miltiadis Allamanis, Earl T Barr, Premkumar Devanbu, and Charles Sutton. 2018. A survey of machine learning for big code and naturalness. *ACM Computing Surveys (CSUR)* 51, 4 (2018), 1–37.

[4] Bandar Aloraini, Massimiliano Menarini, and William G. Griswold. 2019. An Empirical Study of Security Warnings from Static Analysis Tools. *Journal of Systems and Software* 158 (2019), 110414. doi:10.1016/j.jss.2019.110414

[5] Gabor Antal and Sebastian Banescu. 2021. Challenges in benchmarking static analysis tools. In *Proceedings of the 2021 IEEE/ACM 1st International Conference on AI Engineering*.

[6] Farid Binbeshr and Muhammad Imam. 2025. Comparative Analysis of AI-Driven Security Approaches in DevSecOps: Challenges, Solutions, and Future Directions. *arXiv preprint arXiv:2504.19154* (2025). doi:10.48550/arXiv.2504.19154

[7] Valentina Casola, Alessandra De Benedictis, Carmine Mazzocca, and Vincenzo Orbinato. 2024. Secure Software Development and Testing: A Model-Based Approach Integrating Safety and Security for Industry 4.0. *Computers & Security* 137 (2024), 103639. doi:10.1016/j.cose.2023.103639

[8] Brian Chess and Jacob West. 2007. *Secure Programming with Static Analysis*. Addison-Wesley Professional, Boston.

[9] Roland Croft, Dominic Newlands, Ziyu Chen, and M. Ali Babar. 2021. An Empirical Study of Rule-Based and Learning-Based Approaches for Static Application Security Testing. In *Proceedings of the 15th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 1–12. doi:10.1145/3475716.3475781

[10] Fred D Davis. 1989. Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS quarterly* (1989), 319–340.

[11] AR Gray and RB Rainer Jr. 2003. Quantitative and qualitative analysis of factors affecting software process. *Empirical Software Engineering* 8 (2003), 293–306.

[12] Emanuele Iannone, Roberta Guadagni, Filomena Ferrucci, Andrea De Lucia, and Fabio Palomba. 2023. The Secret Life of Software Vulnerabilities: A Large-Scale Empirical Study. *IEEE Transactions on Software Engineering* 49, 1 (2023), 44–63. doi:10.1109/TSE.2022.3140868

[13] Frank Li et al. 2025. Understanding Industry Perspectives of Static Application Security Testing (SAST) Evaluation. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*. 1–12.

[14] Kaixuan Li, Sen Chen, Yue Fang, et al. 2023. Comparison and Evaluation on Static Application Security Testing Tools. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*. ACM, 596–608.

[15] Zhen Li, Deqing Zou, Shouhuai Xu, Hai Jin, and Yawei Zhu. 2018. VulDeePecker: A deep learning-based system for vulnerability detection. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*.

[16] Christopher D Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to information retrieval*. Vol. 1. Cambridge University Press.

[17] Robert C Martin. 2017. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Prentice Hall.

[18] Meta Platforms, Inc. 2020. React - A JavaScript library for building user interfaces. https://reactjs.org/. Acessado em Dez. 2025.

[19] Microsoft Corporation. 2020. TypeScript Language. https://www.typescriptlang.org/. Acessado em Dez. 2025.

[20] Dionas Luan Müller, Matheus Lemes Fialho, Elder de Macedo Rodrigues, and Maicon Bernardino. 2026. Aplicando Ferramentas SAST com IA Gratuitas em uma Ferramenta de Ensino de Gestão de Projetos - Um Benchmarking - Dataset. https://doi.org/10.5281/zenodo.21866721. doi:10.5281/zenodo.21866721 11th Brazilian Symposium on Systematic and Automated Software Testing (SAST 2026), São Paulo, SP, Brazil.

[21] Node.js Foundation. 2020. Node.js - JavaScript runtime built on Chrome's V8 JavaScript engine. https://nodejs.org/. Acessado em Dez. 2025.

[22] P. Nunes, I. Medeiros, J. Fonseca, and M. Vieira. 2018. Benchmarking static analysis tools for web security. In *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*. IEEE, 1–8.

[23] William L Oberkampf and Timothy Guy Trucano. 2006. *Design of and Comparison with Verification and Validation Benchmarks*. Technical Report. Sandia National Lab.(SNL-NM), Albuquerque, NM (United States).

[24] Omitted et al. 2025. Omitted. *arXiv preprint arXiv:omitted* (2025).

[25] Hammond Pearce, Baleegh Ahmad, Benjamin Tan, Brendan Dolan-Gavitt, and Ramesh Karri. 2022. Asleep at the keyboard? Assessing the security of GitHub Copilot's generated code. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 754–768.

[26] PMI. 2021. *A Guide to the Project Management Body of Knowledge (PMBOK Guide)* (7th ed.). Project Management Institute, Newtown Square, PA, USA.

[27] Riccardo Scandariato, James Walden, Aram Hovsepyan, and Wouter Joosen. 2014. Predicting vulnerable software components via text mining. *IEEE Transactions on Software Engineering* 40, 10 (2014), 993–1006.

[28] Gabriel Sousa and Rodrigo Bonifácio. 2024. Mapeamento Sistemático de Comparativos de Ferramentas de Análise Estática de Segurança de Aplicações. In *Anais Estendidos do XXIV Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais*. 281–288.

[29] Benjamin Steenhoek, Md Mahbubur Rahman, Richard Jiles, and Wei Le. 2023. An empirical study of deep learning models for vulnerability detection. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2237–2248.

[30] Stanley Smith Stevens. 1946. On the theory of scales of measurement. *Science* 103, 2684 (1946), 677–680.

[31] Gail Sullivan and Anthony Artino. 2013. Analyzing and Interpreting Data From Likert-Type Scales. *Journal of graduate medical education* 5 (12 2013), 541–2. doi:10.4300/JGME-5-4-18

[32] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in software engineering*. Springer Science & Business Media.

[33] Xiaofan Zhao, Tony Clear, and Ramesh Lal. 2024. Identifying the Primary Dimensions of DevSecOps: A Multi-vocal Literature Review. *Journal of Systems and Software* 214 (2024), 112063. doi:10.1016/j.jss.2024.112063