Fusão de Características em Espaços Vetoriais para Detecção Estática de Cryptojacking em WebAssembly

Bento P. Ch. Baptista[1], Euler Vieira[1,2], Andrés D. Peralta[1], Eduardo L. Feitosa[1]

[1]Instituto de Computação – Universidade Federal do Amazonas (UFAM)
CEP 69.077-000 – Manaus – AM – Brazil

[2]Instituto Federal de Educação, Ciência e Tecnologia do Amazonas (IFAM)
CEP 69.020-120 – Manaus – AM – Brasil

{bento.baptista, eulervieira, andres, efeitosa}@icomp.ufam.edu.br

Abstract. *The proliferation of WebAssembly (Wasm) has fostered cryptojacking campaigns whose opaque binaries evade conventional detection. This paper proposes a static-analysis method based on feature fusion in vector spaces: the binary's Abstract Syntax Tree is segmented into functional chunks, and semantic embeddings are fused with deterministic structural metrics associated with Proof-of-Work routines. Three independent strategies operate over the resulting vector base: topological similarity, supervised learning, and zero-shot inference via LLM. Evaluation on 74 binaries totaling 53,888 chunks showed that aggregated classification via Linear SVM achieves an F1-Score of 0.95, while topological triage attains an MRR of 0.92 with sub-second latency invariant to binary size, enabling both large-scale synchronous triage and asynchronous forensic auditing without dynamic instrumentation.*

Resumo. *A proliferação do WebAssembly (Wasm) tem impulsionado campanhas de cryptojacking cujos binários opacos burlam a detecção convencional. Este artigo propõe um método de análise estática baseado na fusão de características em espaços vetoriais: a Árvore de Sintaxe Abstrata do binário é segmentada em fragmentos funcionais (chunks), e embeddings semânticos são fundidos a métricas estruturais determinísticas associadas a rotinas de Proof-of-Work. Três estratégias independentes operam sobre a base vetorial resultante: similaridade topológica, aprendizado supervisionado e inferência zero-shot via LLM. A avaliação em 74 binários, totalizando 53.888 fragmentos, demonstrou que a classificação agregada via SVM Linear atinge um F1-Score de 0,95, enquanto a triagem topológica alcança um MRR de 0,92 com latência sub-segundo e invariante ao tamanho do binário, permitindo tanto a triagem síncrona em larga escala quanto a auditoria forense assíncrona, sem a necessidade de instrumentação dinâmica.*

1. Introdução

Desde a introdução de serviços de mineração baseados em navegadores em 2017, observa-se uma evolução contínua das técnicas de *cryptojacking*, a mineração não autorizada de criptomoedas utilizando recursos de terceiros, consolidando-as como ameaças persistentes à segurança no ecossistema web [Konoth et al. 2018]. Dados atuais comprovam a gravidade do cenário. No primeiro semestre de 2023, registrou-se um incremento de

400% nos incidentes associados a essa prática [Sonic Wall 2024], com campanhas ativas identificadas em mais de 3.500 domínios [Cside 2025].

Por outro lado, os mecanismos convencionais de detecção enfrentam obstáculos estruturais. Assinaturas estáticas são neutralizadas por ofuscação e diversificação polimórfica de código; a detecção por anomalias de consumo de CPU é enganada mediante *throttling* ou execução intermitente, táticas que mimetizam perfis de uso legítimo; e a análise dinâmica comportamental exige instrumentação, o que introduz *overhead* operacional proibitivo e eleva a taxa de falsos positivos ao processar aplicações legítimas de alta demanda computacional.

Além disso, a disseminação do *WebAssembly* (*Wasm*), um formato binário de baixo nível concebido para execução de alto desempenho em navegadores, introduziu novos vetores de complexidade à inspeção de segurança [Harnes and Morrison 2024b]. A opacidade intrínseca do *Wasm* tem sido explorada para a implementação de mineradores, o que mitiga a eficácia da análise sintática tradicional. Embora ferramentas voltadas à extração de grafos de fluxo de controle (CFG) apresentem resultados promissores na classificação de intenções, tais métodos tendem a demandar alto custo computacional e demonstram vulnerabilidade a ofuscações aplicadas diretamente no *bytecode* [Romano et al. 2020].

Diante desse cenário, este artigo propõe um método de análise estática fundamentado na fusão de características em espaços vetoriais. A abordagem combina, em um único vetor, representações semânticas latentes obtidas a partir da Árvore de Sintaxe Abstrata (AST) do binário com métricas estruturais determinísticas associadas à carga aritmética típica de algoritmos de mineração baseados em *Proof-of-Work* (PoW).

A representação contínua produzida pela fusão é, em si, o produto central do método, não acoplada a uma escolha particular de classificador; os LLMs atuam primariamente como extratores de características contextuais, e não como mecanismos de decisão obrigatórios. Sobre o espaço vetorial resultante operam, de forma independente, três estratégias: similaridade topológica via k-NN, classificadores supervisionados clássicos e inferência semântica *zero-shot* por LLM. Esta separação entre representação e decisão permite instanciar o método com diferentes mecanismos conforme os requisitos operacionais, da triagem síncrona em larga escala à auditoria forense assíncrona, reutilizando a representação vetorial entre eles.

As principais contribuições deste trabalho compreendem: a proposição de um método de análise de código *Wasm* fundamentado na projeção de dados estruturais e semânticos em um espaço vetorial comum; e o desenvolvimento de um *pipeline* analítico que prescinde de execução dinâmica, reduzindo drasticamente os custos de instrumentação e o risco de evasão em tempo de execução. O restante do artigo está organizado da seguinte forma. A Seção 2 apresenta a fundamentação teórica sobre *WebAssembly*, *cryptojacking* e representações de código. A Seção 3 discute os trabalhos relacionados e posiciona a proposta frente ao estado da arte. A Seção 4 detalha o método proposto e a Seção 5 descreve o protocolo experimental. A Seção 6 apresenta os resultados, a telemetria de latência e a discussão dos compromissos entre os mecanismos de classificação. Por fim, a Seção 7 sintetiza as conclusões, as limitações e os trabalhos futuros.

2. Fundamentação Teórica

O *WebAssembly* é um formato de instruções binárias para uma máquina virtual baseada em pilha, projetado para atuar de forma sinérgica ao JavaScript e executar código compilado de linguagens como C, C++ e Rust com desempenho próximo ao nativo [Hoffman 2019, Haas et al. 2017]. Sua segurança fundamenta-se na validação estrutural do artefato, que assegura a integridade do fluxo de controle e a conformidade das operações de memória, e no isolamento em *sandbox*, que restringe o acesso aos recursos do sistema hospedeiro [Perrone and Romano 2025]. Contudo, a opacidade do formato binário e o baixo nível de suas instruções impõem barreiras à inspeção sintática convencional, permitindo a ocultação de rotinas computacionalmente intensivas e potencialmente maliciosas [Romano et al. 2020]. A adoção do *Wasm* por agentes maliciosos é motivada pela eficiência na execução de rotinas matemáticas, significativamente superior a implementações legadas em JavaScript, e pela maior resiliência a detectores baseados em assinaturas textuais ou análise superficial [Konoth et al. 2018].

O *cryptojacking* é a execução não autorizada de algoritmos de mineração de criptomoedas em ativos computacionais de terceiros, tipicamente pela resolução de funções de *hash* sob protocolos de PoW [Konoth et al. 2018, Romano et al. 2020]. A prática está intrinsecamente ligada a criptomoedas orientadas à privacidade, como Monero, cujos algoritmos, a exemplo do RandomX, resistem a *hardware* especializado do tipo ASIC mediante uso intensivo de memória e de operações aritméticas de 64 *bits* [Musch et al. 2019], o que viabiliza a mineração em CPUs de uso geral e transforma computadores domésticos em alvos lucrativos. Cibercriminosos passaram a compilar essas rotinas matemáticas em *WebAssembly*, injetando-as silenciosamente em páginas *web* com velocidade de execução quase nativa; para mitigar a detecção comportamental, mineradores modernos incorporam *throttling*, ajustando dinamicamente a carga de CPU para permanecer abaixo dos limiares de alerta dos sistemas de monitoramento [Sonic Wall 2024].

A inspeção de binários *Wasm* exige a transposição do código para representações estruturais, predominantemente a AST, que detalha a hierarquia das instruções, e o CFG, que mapeia as trajetórias de execução [Romano et al. 2020]. A eficácia dessas estruturas é limitada por técnicas de ofuscação que alteram a morfologia do código sem degradar sua semântica funcional. Neste contexto, o aprendizado de representações propõe o mapeamento de elementos de código para vetores contínuos em espaços de alta dimensionalidade, denominados *embeddings*. Mediante mecanismos de atenção, típicos da arquitetura *Transformer*, torna-se possível capturar dependências contextuais entre instruções não contíguas no binário [Harnes and Morrison 2024b], de modo que a similaridade entre amostras passa a ser quantificada por métricas geométricas, como a similaridade de cosseno, e a detecção de ameaças pode ser tratada como um problema de separação topológica em bases de dados vetoriais [Pan et al. 2023].

3. Trabalhos Relacionados

A detecção de *cryptojacking* no ecossistema web transita desde heurísticas comportamentais baseadas em JavaScript até análises focadas no formato binário do *WebAssembly*, com a literatura organizando-se em dois eixos principais: o perfilamento dinâmico comportamental e microarquitetural e a reconstrução semântica estática [Perrone and Romano 2025]. As primeiras defesas focavam no comportamento

de *scripts* interpretados. O CMTracker [Hong et al. 2018] introduziu heurísticas de rastreamento baseadas em perfiladores iterativos, como frequência de *hashing* e pilhas de chamadas, enquanto o Outguard [Kharraz et al. 2019] adotou um classificador SVM alimentado por métricas do motor *web*, como o número de *WebWorkers* instanciados e o volume de conexões via *WebSockets*. Embora eficazes no auge das campanhas baseadas em JavaScript, estas ferramentas colapsam perante módulos *Wasm*, cuja execução em memória linear oculta as chamadas algorítmicas do inspetor do DOM.

Para contornar a opacidade do *bytecode*, o MineSweeper [Konoth et al. 2018] desce ao nível do processador, monitorando Contadores de Desempenho de Hardware para rastrear anomalias de *cache* e de predição de saltos induzidas pelo algoritmo CryptoNight. Embora a telemetria na CPU seja invulnerável à ofuscação estrutural, o *overhead* da amostragem contínua prejudica a fluidez da navegação, e aplicações *Wasm* legítimas de alto processamento, como renderizadores WebGL e criptografia lícita, geram assinaturas microarquiteturais idênticas às da mineração, elevando os falsos positivos.

Buscando eliminar o *overhead* dinâmico, arquiteturas recentes focaram na análise estática. O MINOS [Naseem et al. 2021] aplica Redes Neurais Convolucionais sobre imagens geradas a partir das matrizes de *bytes* do binário, convertendo a detecção em um problema de visão computacional. Contudo, a diversificação binária automática [Cabrera-Arteaga et al. 2023] destrói a eficácia preditiva de modelos visuais e de assinaturas rasas: a injeção de código morto e o achatamento do fluxo de controle alteram drasticamente a topologia do binário sem penalizar a rotina de mineração. Já o MinerRay [Romano et al. 2020] eleva o *Wasm* a uma Representação Intermediária e constrói exaustivamente um CFG interprocedural, sobre o qual aplica fatiamento de programa para identificar sequências iterativas de *hashing*. Apesar da elevada precisão analítica, o custo de reconstruir e percorrer grafos a partir de binários densos restringe severamente a escalabilidade, inviabilizando a ferramenta como filtro interceptador de baixa latência.

Desenvolvimentos recentes reforçam a corrida armamentista entre detecção e evasão no ecossistema *Wasm*: no eixo ofensivo, ofuscadores dedicados ao formato, como o WASMixer [Cao et al. 2024], e estudos sistemáticos de evasão [Harnes and Morrison 2024a] demonstraram que transformações no *bytecode* degradam severamente detectores de assinaturas visuais e superficiais; no eixo defensivo, o WasmGuard [Peng and outros 2025] propõe a detecção diretamente sobre o binário bruto, com ênfase em resiliência a ofuscação, evidenciando a tendência da área em direção a representações que preservem invariantes semânticas do código.

Em síntese, a literatura expõe um claro *trade-off*: a solução baseada em grafos do MinerRay carece de escalabilidade temporal, a telemetria de *hardware* do MineSweeper impõe *overhead* inaceitável, e as representações superficiais do MINOS e do Outguard sucumbem à diversificação *Wasm* [Perrone and Romano 2025]. O método aqui proposto dispensa a construção morosa de grafos globais e concatena a representação semântica com métricas determinísticas intransponíveis, de modo que o ruído estrutural injetado por ofuscadores resulta apenas em flutuação escalar marginal no hiper-espaço.

4. Método Proposto

O método proposto consiste em um *pipeline* estático e modular que transforma binários *WebAssembly* em vetores passíveis de classificação sobre uma base vetorial, sem qualquer

execução dinâmica do código analisado. A Figura 1 ilustra o fluxo entre os quatro estágios sequenciais (Coleta, Segmentação, Indexação e Classificação) e a interação com a base que centraliza as representações intermediárias. O detalhamento operacional (linguagens, bibliotecas, modelos pré-treinados, hiperparâmetros e infraestrutura) é apresentado na Seção 5, isolando as decisões metodológicas das escolhas tecnológicas.

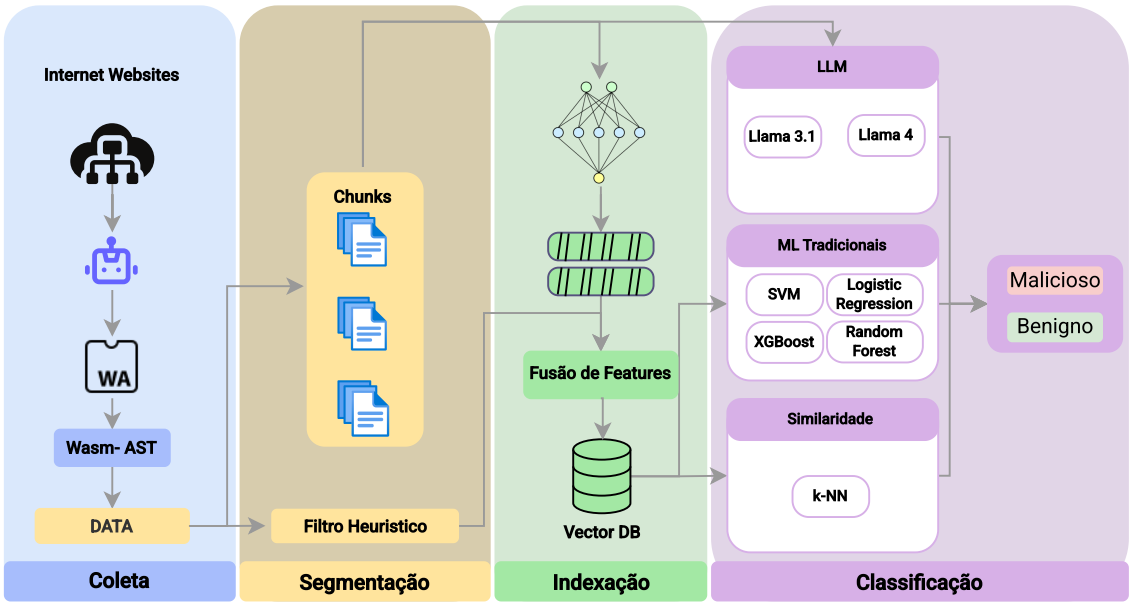

Figura 1. Arquitetura do método proposto, com as quatro etapas e o fluxo de dados desde a interceptação na rede até a classificação.

A primeira etapa, representada no módulo azul da figura, é responsável pela coleta de artefatos *WebAssembly* distribuídos em páginas *web* ativas. Ela parte da premissa estrutural de que mineradores *Wasm* raramente são entregues como artefatos isolados na primeira requisição; tipicamente, são injetados por *scripts* JavaScript que atuam como carregadores e orquestram, em tempo de execução, o *download* e a instanciação do binário. Por essa razão, a coleta opera em duas frentes complementares: a primeira monitora continuamente as requisições assíncronas emitidas pelos *scripts* carregados, interceptando a carga útil secundária na qual o binário habitualmente trafega; a segunda inspeciona o fluxo de dados de forma agnóstica à URL, em busca dos *magic bytes* `\x00asm` característicos do formato, capturando binários servidos fora dos canais convencionais. Após a interceptação, cada binário é desserializado e convertido em sua respectiva AST, expondo a hierarquia das instruções e a topologia das estruturas de controle.

Em seguida, na etapa de segmentação, módulo amarelo da figura, ocorrem dois processamentos em paralelo. No primeiro, as ASTs são segmentadas em unidades funcionais independentes (*chunks*), correspondentes às funções declaradas no módulo *Wasm*, pois o volume típico dos binários inviabiliza o processamento global da árvore por modelos de janela limitada. Sobre cada *chunk* realizam-se duas operações: (i) a planificação da subárvore em uma sequência pseudo-textual processável por modelos *Transformer*, preservando a hierarquia dos blocos de controle e a precedência das instruções por indentação sintética sob percurso DFS; e (ii) a contagem de primitivas estruturais associadas à carga aritmética típica de PoW, em particular operações aritméticas de 64 *bits*, operações *bitwise* (`xor`) e estruturas iterativas explícitas (`loop`).

O segundo é a filtragem heurística de densidade computacional, que descarta subgrafos cuja carga aritmética não atinge um limiar mínimo τ e que não contêm estruturas iterativas explícitas. A fundamentação é estrutural: algoritmos de PoW impõem um volume intransponível de primitivas matemáticas de 64 *bits* para que a mineração seja economicamente viável. A calibração empírica fixa $\tau = 15$, pois rotinas de *hashing* criptográfico superam invariavelmente essa barreira, enquanto funções utilitárias triviais colapsam abaixo de 5 operações. Trabalhos anteriores [Romano et al. 2020, Xia et al. 2023, Cabrera-Arteaga et al. 2023] corroboram que essa carga aritmética constitui invariante semântica persistente sob ofuscação morfológica usual.

A terceira etapa concentra a inovação central do método: a fusão antecipada de duas representações distintas de cada *chunk* em um único vetor, indexado em uma base vetorial, como mostrado no módulo verde. Cada *chunk* funcional é projetado em um espaço $\mathbf{v}_{fusion} \in \mathbb{R}^{d+3}$ pela concatenação de dois componentes complementares: um vetor semântico $\mathbf{v}_{sem} \in \mathbb{R}^d$, obtido pela projeção da AST planificada em um espaço latente por um modelo *Transformer* pré-treinado, capturando dependências semânticas entre instruções não contíguas; e um vetor estrutural $\mathbf{v}_{str} \in \mathbb{R}^3$, composto pelas contagens determinísticas de `i64`, `xor` e `loop`, que injeta no espaço uma assinatura aritmética intransponível para algoritmos de PoW. Formalmente, denotando por $\|$ o operador de concatenação estrita, $\mathbf{v}_{fusion} = [\, \mathbf{v}_{sem} \;\|\; \mathbf{v}_{str} \,]$.

Para evitar que a alta dimensionalidade do componente semântico neutralize a contribuição do componente estrutural durante o cálculo de similaridade, ambos os vetores são normalizados pela norma L_2 antes da concatenação. A motivação da fusão é deliberada: o componente semântico isolado é suscetível a ofuscações superficiais que perturbam a topologia textual da AST, enquanto o componente estrutural isolado é estatisticamente pobre, com apenas três dimensões; a concatenação após normalização garante que a perturbação adversarial sobre um dos componentes resulte apenas em flutuação marginal no hiper-espaço. Os vetores resultantes são indexados em uma base vetorial organizada por um índice de vizinhança aproximada do tipo *Hierarchical Navigable Small World* (HNSW) [Malkov and Yashunin 2018], viabilizando consultas em tempo sub-linear, com similaridade do cosseno como métrica de proximidade, cuja invariância à magnitude mitiga distorções causadas pelas diferentes extensões dos *chunks* [Taipalus et al. 2024]. A cada vetor indexado associa-se um *payload* estruturado contendo o rótulo de classe, o identificador do binário de origem e as contagens estruturais determinísticas isoladas, viabilizando auditoria forense posterior.

Por fim, a etapa de classificação opera sobre as representações indexadas e admite três estratégias complementares e independentes, distintas pela natureza algorítmica, pelo nível de granularidade e pelo perfil operacional. A primeira, Similaridade, é não-paramétrica e opera no nível do *chunk*, empregando k-Nearest Neighbors (k-NN) sobre similaridade do cosseno [Taipalus et al. 2024, Pan et al. 2023] para determinar a classe do fragmento via voto majoritário entre os k vizinhos topológicos, com desempate por similaridade acumulada.

A segunda, ML Tradicional, é supervisionada e paramétrica, operando no nível do binário: os vetores de todos os *chunks* de um mesmo artefato são consolidados via *Mean Pooling*, $\mathbf{v}_{bin} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{v}_{fusion}^{(i)}$, e sobre esse vetor agregado operam classificadores como SVM, *Logistic Regression*, *Random Forest* e XGBoost. A justificativa para essa

agregação é estrutural: o *cryptojacking* se manifesta sistemicamente como um fenômeno de densidade aritmética distribuída, e não como um fragmento isolado.

A terceira, Inferência Semântica por LLM, é não-paramétrica e opera no nível do *chunk*: um Modelo de Linguagem recebe a AST planificada acompanhada de sua telemetria estrutural e emite um veredito binário em modo *zero-shot*, mediado por engenharia de *prompt*. A agregação ao nível do binário segue uma política conservadora de tolerância zero: a detecção de um único *chunk* malicioso implica a classificação punitiva do artefato como um todo, mitigando evasão por diluição sintática típica de campanhas polimórficas. As três estratégias não competem entre si; os compromissos operacionais de cada uma são analisados na Seção 6.

5. Protocolo Experimental

5.1. Implementação, Corpus e Ambiente

A implementação combina duas tecnologias. A linguagem Rust é empregada nas rotinas críticas de desserialização e construção da AST, via biblioteca `walrus`[1], com cada artefato processado em *thread* autônoma e limite estrito de pilha de 64 MB como proteção contra esgotamento de pilha em binários adversarialmente profundos. A linguagem Python opera nas demais etapas, com o rastreador automatizado implementado sobre a biblioteca *Playwright*[2], a base vetorial Qdrant [QDRANT 2026] para armazenamento e recuperação topológica das representações, e a biblioteca *scikit-learn* [Pedregosa et al. 2011] como infraestrutura do *pipeline* de aprendizado supervisionado.

O conjunto de avaliação consistiu em 74 binários *WebAssembly* nativos: 26 aplicações legítimas e 48 artefatos de *cryptojacking*. Os binários maliciosos foram obtidos dos acervos públicos dos projetos MinerRay [Romano et al. 2020] e MineSweeper [Konoth et al. 2018], contemplando desde primitivas clássicas baseadas em *Coinhive* até instrumentações modernas com ofuscação de fluxo de controle. Os benignos foram coletados do catálogo *Made with WebAssembly*[3], complementado por curadoria manual de aplicações de alta densidade computacional, como motores de renderização WebGL e bibliotecas científicas, garantindo à classe legítima complexidade algorítmica comparável à das rotinas maliciosas.

Após a filtragem heurística, o corpus analítico resultou em 53.888 *chunks*, distribuídos de forma fortemente assimétrica: 51.490 benignos e 2.398 maliciosos. Esta assimetria não decorre de viés de amostragem, e sim de característica típica do domínio: motores de *cryptojacking* concentram a carga computacional em poucas funções densas, ao passo que aplicações benignas distribuem a complexidade por um número substancialmente maior de rotinas. A integridade determinística dos ensaios foi garantida pela fixação de semente aleatória global (`random_state=42`) em todos os algoritmos [Géron 2022]. A extração estrutural e a indexação na base vetorial foram executadas em um processador Intel Core Ultra 7-255HX com 32 GB de memória. A inferência por LLM foi processada em uma GPU NVIDIA Quadro RTX 8000 com 48 GB de VRAM.

[1] `https://crates.io/crates/walrus`
[2] `https://playwright.dev/`
[3] `https://madewithwebassembly.com/`

5.2. Protocolo para Similaridade (k-NN)

A avaliação topológica é realizada diretamente sobre o índice HNSW da base vetorial, com $k = 5$ e similaridade do cosseno, via interface `query_points()` do Qdrant, atribuindo a classe por voto majoritário entre os vizinhos recuperados, com desempate por similaridade acumulada e sem etapa de treinamento. O vetor semântico $\mathbf{v}_{sem}$ é avaliado sobre dois modelos pré-treinados: *all-mpnet-base-v2*[4], com $d = 768$, janela de 512 *tokens* e lotes de 128 *chunks*, e *Qwen3-Embedding-0.6B*[5], com $d = 1024$, janela de 2048 *tokens* e lotes de 4 *chunks* em razão da pegada de VRAM. Após a concatenação com o vetor estrutural, os tensores finais possuem 771 e 1027 dimensões, persistidos em coleções distintas para fins comparativos. A consolidação por *chunk* adota *Mean Pooling* sobre os estados ocultos da última camada, garantindo invariância à extensão do fragmento. Os parâmetros do índice HNSW seguem [Malkov and Yashunin 2018]: $m = 16$ e *ef_construct* $= 100$.

5.3. Protocolo para ML Tradicional

Os classificadores supervisionados operam sobre a representação agregada do binário via *Mean Pooling* dos vetores de seus *chunks*. A normalização por `StandardScaler` é encapsulada em um *pipeline*, calibrada exclusivamente sobre o subconjunto de treino de cada *fold* para prevenir vazamento de dados. A avaliação adota validação cruzada estratificada 5-*fold*, complementada por particionamento *hold-out* estratificado 80/20 para teste em amostras não vistas. Os hiperparâmetros foram fixados a priori: SVM com *kernel* linear e $C = 0.5$; *Logistic Regression* com regularização L_2, $C = 0.5$ e *max_iter* $= 2000$; *Random Forest* com *n_estimators* $= 100$ e *max_depth* $= 3$; e XGBoost com *learning_rate* $= 0.05$ e *max_depth* $= 2$. A opção por valores conservadores em detrimento de busca em grade é deliberada e responde ao regime de baixa amostragem, no qual a otimização agressiva tende a inflar artificialmente as métricas e comprometer a generalização externa.

5.4. Protocolo para LLM

A estratégia de inferência semântica emprega dois modelos abertos, instanciados localmente via Ollama e quantizados em 4 *bits* (*Q4_K_M*): *Llama 3.1 8B Instruct* e *Llama 4 Dolphin 8B Instruct*. A concorrência de sessões é gerida por um motor assíncrono baseado em `aiohttp`, e ambos os modelos operam com temperatura $T = 0.1$ e $\text{top}_p = 0.9$ para favorecer respostas determinísticas. A interação segue uma estratégia de *Zero-Shot System Role-Play*, na qual o modelo é instruído a atuar como analista sênior de *malware* especializado em binários de baixo nível, recebendo a AST planificada e a telemetria estrutural de cada *chunk*. O *prompt* impõe três regras de julgamento: não classificar operações *bitwise* como maliciosas sem telemetria anômala, dado seu uso legítimo em criptografia e processamento gráfico; avaliar ciclos de alta repetição à luz dos metadados comportamentais; e adotar o veredito benigno como padrão para computação de alta complexidade sem telemetria suspeita. A estrutura completa do *prompt* integra os artefatos públicos do trabalho.

6. Resultados e Discussão

A Tabela 1 consolida o desempenho preditivo ao nível do *chunk* para os motores de Similaridade (k-NN) e Inferência por LLM. O espaço latente gerado pelo modelo Qwen

[4]`https://huggingface.co/sentence-transformers/all-mpnet-base-v2`
[5]`https://huggingface.co/Qwen/Qwen3-Embedding-0.6B`

(1027 dimensões) atingiu acurácia global de 94.88% e *Mean Reciprocal Rank* (MRR) de 0.9188. O modelo MPNet registrou acurácia de 90.23% e *recall* de 80.87%.

Tabela 1. Métricas de desempenho *chunk* para o protocolo experimental de Similaridade (k-NN) e Inferência por LLM.

Modelo	Acurácia	Precisão	Recall	MRR
Similaridade (k-NN, $k = 5$)				
Qwen	94.88%	35.02%	72.19%	0.9188
MPNet	90.23%	21.83%	80.87%	0.9027
Inferência por LLM				
Llama 4 Dolphin 8B Instruct	94.19%	41.72%	77.23%	0.7723
Llama 3.1 8B Instruct	82.06%	18.37%	87.99%	0.8799

Observou-se uma compressão estatística na métrica de precisão, de 35.02% para a arquitetura Qwen, fenômeno atribuído à severa assimetria de classes ao nível do *chunk* e ao clássico paradoxo da acurácia: sob razão aproximada de 21:1, qualquer falso positivo é amplificado, deprimindo a precisão mesmo quando a recuperação dos verdadeiros positivos é elevada.

Embora a literatura recomende técnicas de sobreamostragem sintética [Chawla et al. 2002] ou balanceamento de pesos para contornar esse viés, a aplicação de dados artificiais foi deliberadamente descartada nesta fase: a injeção de tensores sintéticos corromperia a densidade topológica natural das representações no banco vetorial, e a manutenção da distribuição original é mandatória para aferir o desempenho em cenário operacional realista, no qual funções legítimas de processamento intensivo predominam e geram vetores topologicamente próximos às rotinas de PoW [Loose et al. 2023]. O elevado MRR comprova a viabilidade deste motor para triagem e auditoria forense, garantindo que os fragmentos maliciosos figurem no topo da lista de recuperação, ainda que o desbalanceamento limite sua aplicação como sistema autônomo de bloqueio em tempo real.

A Tabela 2 apresenta o desempenho dos classificadores supervisionados sobre as representações agregadas via *Mean Pooling* (espaço Qwen), sob validação cruzada estratificada 5-*fold*.

Tabela 2. Métricas globais da validação cruzada 5-*fold* sobre as representações agregadas.

Modelo Estatístico	Acurácia	Precisão	Recall	F1-Score
SVM Linear	0.9314 ± 0.042	0.9303 ± 0.048	0.9778 ± 0.031	0.9501 ± 0.036
Logistic Regression	0.9314 ± 0.042	0.9232 ± 0.051	0.9778 ± 0.031	0.9483 ± 0.038
Random Forest	0.9038 ± 0.056	0.9300 ± 0.062	0.9378 ± 0.058	0.9291 ± 0.054
XGBoost	0.9057 ± 0.052	0.9174 ± 0.059	0.9356 ± 0.055	0.9250 ± 0.051

Nota-se que o desempenho dos classificadores lineares, SVM Linear e *Logistic Regression*, é estatisticamente equivalente, dada a sobreposição substancial dos intervalos de variação. Adotamos o SVM Linear como referência nos relatos subsequentes por

sua leve vantagem nominal, mas a observação metodologicamente relevante é a superioridade sistemática dos modelos lineares sobre os de particionamento hierárquico. Em hiper-espaços de alta dimensionalidade, com 1027 dimensões contínuas, os dados adquirem alta separabilidade linear, condição na qual hiperplanos de margem máxima mitigam o *overfitting* intrínseco às árvores de decisão [Cortes and Vapnik 1995]. A predominância dos lineares é ainda coerente com o regime de baixa amostragem do problema: classificadores de capacidade elevada tendem a se ajustar excessivamente a particularidades do conjunto de treinamento em corpora restritos, ao passo que modelos lineares sobre representações bem-construídas exploram a estrutura geométrica do espaço fundido com viés indutivo apropriado [Cabrera-Arteaga et al. 2023].

A inferência semântica via *prompting Zero-Shot* indicou o modelo *Llama 4 Dolphin 8B Instruct* como o mais adequado, entre os avaliados, para tarefas de inspeção técnica, embora com taxa de falsos positivos ainda incompatível com bloqueio autônomo. A arquitetura conservadora do Llama 3.1 8B Instruct produziu 9.378 falsos positivos em um universo de 12.250 *chunks* benignos de teste, resultando em uma taxa de falsos positivos (FPR) severa de 76.55%. O Llama 4 Dolphin interpretou com maior assertividade as funções de processamento intensivo legítimas, reduzindo os alarmes falsos para 5.805, com FPR de 47.38%, e mantendo recuperação de verdadeiros positivos equivalente, com 41.103 acertos contra 41.355 do predecessor [Ma et al. 2023].

Em termos de telemetria de latência, a busca topológica iterativa na base vetorial obteve tempos médios de 0.22 segundos por binário, ao passo que a inferência por LLM exigiu 0.87 segundos por *chunk*. A independência entre o tempo de inferência e o tamanho do binário foi avaliada pela análise de sensibilidade do Coeficiente de Correlação de Pearson, apresentada na Figura 2: para cenários realistas de dispersão dos tamanhos ($\sigma_S \geq 2$ MB), observa-se $r < 0.0084$, abaixo do limiar de significância para $N = 53.888$ amostras ($\alpha = 0.05$), confirmando que o *overhead* computacional concentra-se no comprimento fixo da representação latente. Logo, a base vetorial deve atuar como interceptador síncrono preliminar, reservando o motor LLM para rotinas assíncronas de *Threat Intelligence*.

No que concerne às etapas anteriores à classificação, a desserialização, a construção da AST e a contagem estrutural são executadas pela camada Rust em tempo proporcional ao tamanho do artefato, com *thread* autônoma e pilha limitada a 64 MB, mantendo o consumo de memória previsível mesmo para binários adversarialmente profundos. A geração dos *embeddings* constitui o custo dominante da indexação, sendo executada uma única vez por artefato e amortizada entre os três mecanismos, que reutilizam a mesma representação sem reprocessamento do binário. A medição sistemática de latência e consumo de recursos de ponta a ponta, incluindo a interceptação no navegador do usuário final, integra os trabalhos futuros.

6.1. Discussão

A convergência dos três experimentos revela um sistema que opera em diferentes níveis de abstração, com compromissos distintos entre sensibilidade analítica e precisão operacional. A classificação ao nível do *chunk* evidenciou as limitações da análise de alta granularidade: embora o motor de similaridade alcance MRR superior a 0.91, a precisão nominal reflete a ambiguidade semântica intrínseca ao *Wasm*, na qual funções legítimas

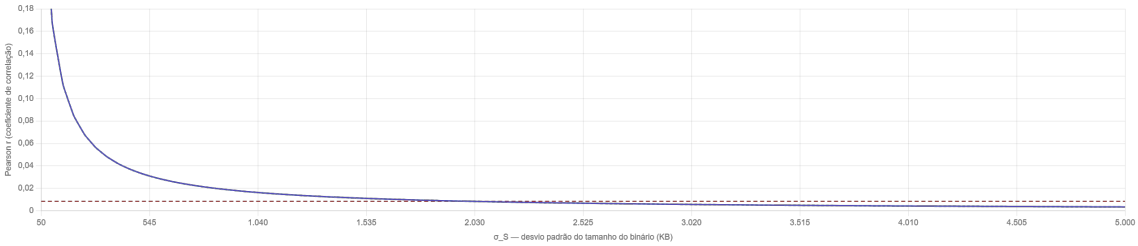

Figura 2. Análise de sensibilidade do Coeficiente de Correlação de Pearson (r) em relação à variância do tamanho original do binário (σ_S).

de renderização gráfica ou criptografia utilitária partilham a geometria de instruções típica dos mineradores. A transição para a análise agregada via *Mean Pooling* demonstrou que a atividade maliciosa se manifesta como fenômeno de densidade estrutural distribuída: a consolidação dos vetores dilui o ruído de funções benignas isoladas e acentua a carga aritmética contínua que caracteriza os mineradores. Por sua vez, a inferência por LLM demonstrou capacidade para resolver ambiguidades matemáticas que confundem os classificadores estatísticos, embora a latência direcione sua utilidade a processos assíncronos.

A Tabela 3 sintetiza os compromissos operacionais entre os três mecanismos. A triagem por similaridade oferece a menor latência e dispensa treinamento, sendo adequada à interceptação síncrona e à priorização de amostras em fluxos de *Threat Intelligence*, nos quais o MRR elevado garante que fragmentos suspeitos figurem no topo da fila de análise; sua fragilidade é a precisão nominal sob desbalanceamento extremo, o que a desaconselha como mecanismo autônomo de bloqueio. A classificação supervisionada agregada apresenta o melhor equilíbrio entre métricas e custo, sendo indicada para varredura em larga escala e bloqueio preventivo; sua principal exposição a evasão é a diluição deliberada de rotinas maliciosas em binários inflados com funções benignas, parcialmente mitigada pela filtragem heurística de densidade. A inferência por LLM é a única capaz de justificar semanticamente o veredito e generalizar para lógicas não observadas no corpus, propriedade valiosa para auditoria forense; seu custo por *chunk* e sua taxa de falsos positivos restringem seu emprego a rotinas assíncronas. Quanto à escalabilidade, os dois primeiros mecanismos operam sobre índice HNSW sublinear e vetores de dimensão fixa, ao passo que o custo do LLM cresce linearmente com o número de *chunks*.

Tabela 3. Compromissos operacionais entre os mecanismos de classificação propostos.

Mecanismo	Granularidade	Latência	Cenário de aplicação	Limitação principal
Similaridade k-NN	*Chunk*	0.22 s por binário	Triagem síncrona e priorização	Precisão sob desbalanceamento
ML supervisionado	Binário	Sub-segundo	Varredura em larga escala e bloqueio	Diluição via *Mean Pooling*
Inferência por LLM	*Chunk*	0.87 s por *chunk*	Auditoria forense e *Threat Intel.*	Latência e falsos positivos

Cabe registrar que os valores reunidos na Tabela 4 foram obtidos sob corpora, protocolos experimentais e condições de avaliação distintos entre si. A tabela cumpre, portanto, papel de posicionamento qualitativo do método frente à literatura, e não de confronto experimental direto, cuja realização exigiria a reexecução das ferramentas sob um protocolo unificado, o que se planeja como trabalho futuro sobre o corpus público disponibilizado. Registre-se ainda que os F1-Scores nominalmente superiores das soluções de referência foram medidos antes da consolidação

Tabela 4. Posicionamento indicativo do método proposto frente a soluções de referência para detecção de *cryptojacking*.

Ferramenta / Autor	Abordagem	F1-Score
MineSweeper [Konoth et al. 2018]	Dinâmica (HPC / CPU)	N/A (TPR 99%)
Outguard [Kharraz et al. 2019]	Dinâmica (SVM / Rede)	0.9700
MinerRay [Romano et al. 2020]	Estática (Grafo ICFG)	0.9800
MINOS [Naseem et al. 2021]	Estática (Visão / CNN)	0.9890
Nossa Abordagem (SVM Linear)	**Estática (Early Fusion)**	**0.9501**

das técnicas modernas de diversificação de *Wasm*, perante as quais a literatura subsequente [Cabrera-Arteaga et al. 2023, Loose et al. 2023] demonstrou colapso severo de analisadores visuais e de fluxo interprocedural, com taxas elevadas de falsos negativos.

O F1-Score de 0.9501 alcançado pela arquitetura de *Early Fusion* reflete, em contrapartida, um desempenho operacional medido em condições adversas, obtido sobre granularidade fina e classes desbalanceadas, prescindindo de instrumentação dinâmica. A resistência ao polimorfismo estrutural decorre diretamente da incorporação de métricas determinísticas ao vetor de fusão, as quais absorvem o ruído adversarial sem custo adicional de latência.

7. Conclusão

Este artigo investigou a detecção de *cryptojacking* em ambientes *web*, com foco nas dificuldades introduzidas pelo formato binário *WebAssembly*, propondo um método de análise estática fundamentado na fusão precoce de *embeddings* semânticos da AST com métricas determinísticas de carga aritmética de PoW, sobre a qual operam três estratégias independentes de classificação.

A avaliação empírica em 74 binários, com 53.888 *chunks*, sustentou três conclusões. Primeiro, a análise isolada de fragmentos padece de ambiguidade semântica intrínseca ao *Wasm*; a agregação via *Mean Pooling* mitiga essa limitação e permite separação linear das classes, com o SVM Linear atingindo F1-Score de 0.9501 ± 0.036 sob validação cruzada 5-*fold*. Segundo, a topologia do hiper-espaço fundido sustenta triagem por similaridade com MRR de 0.92 e latência sub-segundo invariante ao tamanho do binário. Terceiro, a inferência por LLM contribui com discernimento semântico fino, apropriada para processos assíncronos de *threat intelligence* dado o custo de latência de 0.87 s por *chunk*.

Do ponto de vista das implicações práticas, a separação arquitetural entre representação vetorial e mecanismo de decisão é o principal contributo metodológico do trabalho: o vetor constitui um substrato reutilizável sobre o qual diferentes classificadores podem ser instanciados conforme o regime operacional, seja a triagem síncrona em larga escala via base vetorial, seja a auditoria forense assíncrona via LLM, sem reprocessamento do binário. O extrator isolado em Rust, focado em contagens atômicas, resiste a aninhamentos maliciosos geradores de *Stack Overflow*, e técnicas superficiais como minificação ou renomeação têm impacto estatisticamente nulo sobre o vetor estrutural $\mathbf{v}_{str}$. Em síntese, os resultados sustentam a viabilidade do método no regime operacional

para o qual foi concebido, sem qualquer execução dinâmica e com corpus, ASTs e vetores publicados para reprodução independente; as limitações a seguir delimitam o escopo de aplicação de cada mecanismo e orientam a agenda de pesquisa, sem comprometer o núcleo da contribuição.

7.1. Limitações e Trabalhos Futuros

Seis limitações devem ser explicitadas. Primeiro, o corpus restrito de 74 binários limita a generalização estatística externa e reflete a escassez de *datasets* públicos anotados para mineradores *Wasm*; os desvios da validação cruzada quantificam essa incerteza. Segundo, os modelos MPNet e Qwen são empregados sem ajuste fino para *Wasm*, o que pode limitar a fidelidade da representação semântica, ao custo de preservar o baixo custo e a reprodutibilidade do método. Terceiro, o *Mean Pooling* pode diluir trechos maliciosos pontuais em binários com muitas funções benignas; a filtragem de densidade e a política punitiva por *chunk* atuam como contramedidas, mas a inflação deliberada de código benigno permanece em aberto. Quarto, a assinatura estrutural baseia-se em três sinais associados aos esquemas de PoW dominantes; famílias emergentes que desloquem a carga para outras primitivas exigiriam recalibração do vetor estrutural, cuja extensão dimensional é direta por concatenação. Quinto, a projeção textual plana da AST pode ser comprometida por nivelamento de fluxo de controle ou inserção agressiva de predicados opacos, diluindo a densidade de `i64` entre *chunks* artificialmente induzidos. Sexto, a inferência semântica profunda apresenta latência e taxa de falsos positivos incompatíveis com bloqueio estritamente síncrono. Adicionalmente, o corpus e o extrator refletem a especificação consolidada do *Wasm* 1.0: extensões recentes, como o WasmGC, o endereçamento de 64 *bits* da proposta Memory64 e o uso de *threads* condicionado às políticas de isolamento COOP e COEP, alteram o repertório de instruções e os vetores de entrega dos mineradores, demandando reavaliação da assinatura estrutural e da etapa de coleta.

Como trabalhos futuros, planeja-se investigar os limites da arquitetura perante ofuscação polimórfica extrema, a exemplo do *wasm-mutate* modificado, com poda de código morto na camada Rust; expandir um *corpus* de referência aberto, contemplando variantes adversariais e binários com extensões modernas da especificação; realizar o ajuste fino de modelos de representação sobre corpora *Wasm*; reexecutar detectores representativos do estado da arte sob protocolo unificado sobre o corpus publicado, viabilizando a comparação direta ausente na Tabela 4; medir latência e consumo de recursos de ponta a ponta, da interceptação no navegador à decisão; e otimizar a inferência por LLM mediante destilação de modelos especializados e *caching* semântico.

Disponibilidade de artefatos

Os autores declaram que os artefatos de pesquisa que apoiam as descobertas deste estudo, incluindo os dados experimentais, os prompts, o código-fonte e os resultados, estão disponíveis em: `https://github.com/yekwim/wasm_crypto_finder`.

Declaração de Uso de Inteligência Artificial Generativa

Em conformidade com as diretrizes do SBSeg 2026, os autores declaram o uso da ferramenta *Google Gemini* exclusivamente para apoio à revisão de estilo e clareza redacional. A responsabilidade integral sobre o conteúdo, a correção das afirmações técnicas e a integridade dos resultados permanece exclusivamente dos autores.

Agradecimentos

Os autores agradecem ao Programa de Pós-Graduação em Informática da Universidade Federal do Amazonas (PPGI/UFAM) e ao Instituto Federal do Amazonas (IFAM). Esta pesquisa foi parcialmente financiada, conforme previsto nos Arts. 21 e 22 do Decreto No. 10.521/2020, nos termos da Lei Federal No. 8.387/1991, através do convênio No. 015/2025, firmado entre ICOMP/UFAM, Digiboard da Amazônia Ltda e Lenovo Ltda. Esta pesquisa foi parcialmente financiada pela Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), por meio do PROEX – Código Financeiro 001, e pela Fundação de Amparo à Pesquisa do Estado do Amazonas (FAPEAM), Edital POSGRAD 2024/2025.

Referências

Cabrera-Arteaga, J., Monperrus, M., Toady, T., and Baudry, B. (2023). Webassembly diversification for malware evasion. *Computers & Security*, 131:103296.

Cao, S., He, N., Guo, Y., and Wang, H. (2024). Wasmixer: Binary obfuscation for webassembly. In *Computer Security – ESORICS 2024*. Springer.

Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. (2002). Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3):273–297.

Cside (2025). Cryptojacking is dead: long live cryptojacking. acessado em 15-10-2025.

Géron, A. (2022). *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow*. "O'Reilly Media, Inc.".

Haas, A., Rossberg, A., Schuff, D. L., Titzer, B. L., Holman, M., Gohman, D., Wagner, L., Zakai, A., and Bastien, J. (2017). Bringing the web up to speed with webassembly. *SIGPLAN Not.*, 52(6):185–200.

Harnes, H. and Morrison, D. (2024a). Cryptic bytes: Webassembly obfuscation for evading cryptojacking detection. *arXiv preprint arXiv:2403.15197*.

Harnes, H. and Morrison, D. (2024b). Sok: Analysis techniques for webassembly. *Future Internet*, 16(3):84.

Hoffman, K. (2019). *Programming WebAssembly with Rust: unified development for web, mobile, and embedded applications*. The Pragmatic Bookshelf.

Hong, G., Yang, Z., Yang, S., Zhang, L., Nan, Y., Zhang, Z., Yang, M., Zhang, Y., Qian, Z., and Duan, H. (2018). How you get shot in the back: A systematical study about cryptojacking in the real world. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 1701–1713, New York, NY, USA. Association for Computing Machinery.

Kharraz, A., Ma, Z., Murley, P., Lever, C., Mason, J., Miller, A., Borisov, N., Antonakakis, M., and Bailey, M. (2019). Outguard: Detecting in-browser covert cryptocurrency mining in the wild. In *The World Wide Web Conference*, WWW '19, page 840–852, New York, NY, USA. Association for Computing Machinery.

Konoth, R. K., Vineti, E., Moonsamy, V., Lindorfer, M., Kruegel, C., Bos, H., and Vigna, G. (2018). Minesweeper: An in-depth look into drive-by cryptocurrency mining and its defense. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 1714–1730, New York, NY, USA. Association for Computing Machinery.

Loose, N., Mächtle, F., Pott, C., Bezsmertnyi, V., and Eisenbarth, T. (2023). Madvex: instrumentation-based adversarial attacks on machine learning malware detection. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 69–88. Springer.

Ma, W., Liu, S., Lin, Z., Wang, W., Hu, Q., Liu, Y., Zhang, C., Nie, L., Li, L., and Liu, Y. (2023). Lms: Understanding code syntax and semantics for code analysis. *arXiv preprint arXiv:2305.12138*.

Malkov, Y. A. and Yashunin, D. A. (2018). Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence*, 42(4):824–836.

Musch, M., Wressnegger, C., Johns, M., and Rieck, K. (2019). Thieves in the browser: Web-based cryptojacking in the wild. In *Proceedings of the 14th International Conference on Availability, Reliability and Security*, ARES '19, New York, NY, USA. Association for Computing Machinery.

Naseem, F. N., Aris, A., Babun, L., Tekiner, E., and Uluagac, A. S. (2021). Minos: A lightweight real-time cryptojacking detection system. In *NDSS*.

Pan, J. J., Wang, J., and Li, G. (2023). Survey of vector database management systems. *arXiv preprint arXiv:2310.14021*.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830.

Peng, J. and outros (2025). Wasmguard: Enhancing web security through robust raw-binary detection of webassembly malware. In *Proceedings of the ACM Web Conference 2025 (WWW '25)*. ACM.

Perrone, G. and Romano, S. P. (2025). Webassembly and security: a review. *Computer Science Review*, 56:100728.

QDRANT (2026). *Qdrant: High-performance vector search engine*. Qdrant Technologies. Versão 1.18.

Romano, A., Zheng, Y., and Wang, W. (2020). Minerray: Semantics-aware analysis for ever-evolving cryptojacking detection. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, pages 1129–1140.

Sonic Wall (2024). Sonicwall cyber threat report. https://www.sonicwall.com/threat-report. acessado em 17-Dez-2024.

Taipalus, T., Grahn, H., Turtiainen, H., and Costin, A. (2024). Utilizing vector database management systems in cyber security. In *Proceedings of the 23th European Conference on Cyber Warfare and Security. Academic Conferences International Ltd*.

Xia, Y., He, P., Zhang, X., Liu, P., Ji, S., and Wang, W. (2023). Static semantics reconstruction for enhancing javascript-webassembly multilingual malware detection. In *European Symposium on Research in Computer Security*, pages 255–276. Springer.