UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Matheus Santos

Aprendizado de Máquina e Interpretabilidade com SHAP para Detecção de Spyware em Dados de Memória

Uberlândia, Brasil

2025

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Matheus Santos

Aprendizado de Máquina e Interpretabilidade com SHAP para Detecção de Spyware em Dados de Memória

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Sistemas de Informação.

Orientador: Diego Nunes Molinos

Coorientador: Rodrigo Sanches Miani

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Sistemas de Informação

Uberlândia, Brasil

2025

Matheus Santos

Aprendizado de Máquina e Interpretabilidade com SHAP para Detecção de Spyware em Dados de Memória

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Sistemas de Informação.

Trabalho aprovado. Uberlândia, Brasil, 15 de maio de 2025:

Diego Nunes Molinos
Orientador

Claudiney Ramos Tinoco
Professor

Thiago Pirola Ribeiro
Professor

Uberlândia, Brasil
2025

Agradecimentos

Agradeço primeiramente a Deus, pela força e determinação ao longo desta jornada. À minha família, que sempre esteve ao meu lado, oferecendo apoio e inspiração. À minha companheira, pelo amor, compreensão e incentivo constante, tornando cada desafio mais leve e cada conquista ainda mais especial. Aos meus orientadores e professores, cuja orientação e conhecimento foram essenciais para a realização deste trabalho. Aos amigos e colegas, pelo suporte, discussões enriquecedoras e momentos de aprendizado compartilhados. A todos que — direta ou indiretamente — contribuíram para que este trabalho se tornasse realidade, minha sincera gratidão.

"A educação é o bem mais poderoso que você pode usar para mudar o mundo." —
Nelson Mandela

Resumo

A segurança da informação enfrenta desafios cada vez maiores com o avanço de ameaças cibernéticas sofisticadas, como os *spywares*. Esses *malwares* atuam de forma silenciosa, dificultando sua detecção por soluções tradicionais baseadas em assinaturas. Este trabalho propõe o desenvolvimento de um algoritmo baseado em aprendizado de máquina para a detecção de *spyware* por meio da análise de memória volátil, utilizando o conjunto de dados **CIC-MalMem-2022**. Foram aplicadas técnicas de **pré-processamento**, **seleção de atributos** e **validação cruzada** para treinar e avaliar quatro modelos: **Árvore de Decisão**, **Naive Bayes**, ***Support Vector Machine (SVM)*** e **Rede Neural**. A avaliação dos modelos considerou métricas como acurácia e tempo de inferência. Os resultados demonstraram que é possível identificar comportamentos maliciosos relacionados a *spywares* com elevada precisão utilizando conjuntos de dados extraídos da memória. Este estudo contribui para o avanço de abordagens inteligentes na segurança cibernética, explorando um tipo de ameaça pouco abordado em relação a outros tipos de *malware*.

Palavras-chave: *Malware*, *Spyware*, Segurança da Informação, *Machine Learning*, Inteligência Arficial.

Lista de ilustrações

Lista de siglas

API Application Programming Interface

AI Artificial Intelligence

DLL Dynamic Link Library

DeepLIFT Deep Learning Important FeaTures

LIME Local Interpretable Model-agnostic Explanations

ML Machine Learning

RAM Random Access Memory

SVM Support Vector Machine

SHAP SHapley Additive exPlanations

URL Uniform Resource Locators

XAI Explainable Artificial Intelligence

Lista de tabelas

Sumário

1 Introdução

A crescente dependência da sociedade moderna em tecnologias digitais e conectividade tem trazido benefícios significativos, mas também expõe indivíduos e organizações a ameaças cibernéticas cada vez mais sofisticadas. Entre essas ameaças, os *spywares* se destacam por sua capacidade de operar de forma furtiva, monitorando e coletando informações sensíveis sem o consentimento do usuário (ABUALHAJ et al., 2024; ABHIJNA et al., 2024).

Nos últimos anos, o uso de *spywares* comerciais, como o notório Pegasus da NSO Group, tem se expandido globalmente, sendo utilizado não apenas por governos autoritários, mas também por cibercriminosos em diferentes esferas. Esses *softwares* permitem acesso remoto a dispositivos, possibilitando a escuta de chamadas, leitura de mensagens e controle de câmeras e microfones — tudo de forma invisível para o usuário (RICHARD; RIGAUD, 2023).

De acordo com a Cybersecurity Ventures, o custo global do cibercrime foi estimado em US$ 8 trilhões em 2023, com projeções indicando que o valor pode atingir US$ 10,5 trilhões até 2025 (VENTURES, 2022). Essa estimativa inclui diversos tipos de crimes cibernéticos, como roubo de dados, fraudes financeiras, ataques de *ransomware* e, notavelmente, *spywares*. Os *spywares*, por sua vez, representam uma parcela significativa dessas perdas, pois comprometem a privacidade, roubam credenciais e instalam cargas adicionais maliciosas sem o consentimento do usuário (VENTURES, 2022).

No contexto de detecção, a identificação de *spywares* é particularmente desafiadora, pois sua operação dificulta a aplicação de métodos tradicionais baseados em assinaturas ou regras fixas. Apesar da utilização de *softwares* antivírus, *firewalls* e práticas seguras de navegação, os métodos tradicionais de defesa, muitas vezes, se mostram insuficientes diante da rápida evolução e disseminação dos *spywares* (ABUALHAJ et al., 2024).

De acordo com Abualhaj et al. (2024), soluções baseadas em assinaturas, como os antivírus convencionais, falham em detectar até 80% dos *keyloggers*, um tipo comum de *spyware* utilizado para capturar informações sensíveis do usuário. Dentro deste contexto, ressalta-se a importância de abordagens mais sofisticadas, como a análise heurística e, especialmente, o uso de técnicas de Machine Learning (ML), que permitem a detecção proativa de ameaças ao analisar padrões de comportamento e ações suspeitas.

Ao contrário das defesas estáticas, os modelos de ML têm a capacidade de se adaptar continuamente a novas variantes de *malware*, aprendendo com dados anteriores e melhorando sua capacidade de detecção com o tempo.

Sob a óptica do conjunto de dados utilizado pelos algoritmos de ML, a extração de dados da memória Random Access Memory (RAM) tem se mostrado eficaz na detecção de *spyware* devido à sua capacidade de revelar comportamentos maliciosos em tempo real. Segundo Abualhaj et al. (2024), ao utilizar o *dataset* Obfuscated-MalMem2022 — composto por amostras de memória de diferentes famílias de *spywares* — e aplicar algoritmos de aprendizado de máquina, foi possível atingir 99,97% de acurácia na detecção. Ademais, a análise de memória supera abordagens tradicionais ao identificar padrões que escapam de métodos baseados em assinatura, reforçando sua relevância em sistemas de segurança baseados em Artificial Intelligence (AI).

Com base no cenário atual de crescimento das ameaças representadas por *spyware* e nas limitações dos métodos tradicionais de detecção, este trabalho tem como propósito realizar um estudo comparativo entre diferentes algoritmos de aprendizado de máquina, tais como: **Decision Tree, SVM, Naive Bayes e Redes Neurais**, aplicados à detecção de *spyware* a partir de dados extraídos da memória RAM. O objetivo principal desta proposta é avaliar o desempenho preditivo desses modelos com foco na precisão, sensibilidade e robustez frente a dados comportamentais de diferentes famílias de *spyware*. Além da avaliação dos algoritmos, o estudo propõe uma análise de otimização de atributos (*feature selection*), buscando reduzir a dimensionalidade dos dados sem comprometer a capacidade de detecção.

Outra contribuição deste trabalho é a aplicação do método SHapley Additive exPlanations (SHAP) para interpretabilidade dos modelos, permitindo compreender a importância de cada feature para a decisão do classificador. Essa abordagem contribuiu para uma maior transparência e confiabilidade dos sistemas de detecção, ao tornar explícitos os fatores que influenciam a identificação de comportamentos suspeitos na memória.

1.1 Justificativa

O crescimento exponencial da sofisticação dos *malwares* representa um grande desafio para a segurança cibernética. Esses mecanismos são capazes de monitorar as ações do usuário, registrar teclas digitadas, capturar telas e até mesmo enviar informações confidenciais a servidores remotos, sem qualquer aviso perceptível. Além disso, como muitas vezes não apresentam sintomas visíveis, podem permanecer ativos por longos períodos, expondo sistemas e usuários a riscos contínuos (BENSAOUD; KALITA; BENSAOUD, 2024).

Embora existam diversas ferramentas de segurança, a maioria se baseia em assinaturas conhecidas ou em comportamentos previamente catalogados, tornando-se ineficaz contra variantes novas ou modificadas. Nesse sentido, abordagens baseadas em aprendizado de máquina se tornam indispensáveis, pois são capazes de aprender com os dados

e identificar comportamentos suspeitos de forma adaptativa (LEONEL; MOLINOS; MIANI, 2024; NASSER; NASSAR, 2023).

Ante o exposto, a utilização de técnicas forenses baseadas na análise de dados extraídos de memória permite observar diretamente o comportamento dos processos em execução, acessos a bibliotecas dinâmicas *(Dynamic Link Library (DLL))*, alterações em registros e outros indicadores que frequentemente passam despercebidos em análises convencionais (LEONEL; MOLINOS; MIANI, 2024; DENER, 2022). Esta abordagem, juntamente com o advento da inteligência artificial AI amplia a cobertura da detecção e pode ser integrada a sistemas de segurança, contribuindo para uma solução moderna e otimizada.

1.2 Hipótese

Modelos de aprendizado de máquina, treinados com dados extraídos da memória RAM, são capazes de detectar spywares com alta acurácia, bem como a aplicação de técnicas de seleção de atributos, combinadas com interpretabilidade via SHAP, permite reduzir a complexidade do modelo sem comprometer seu desempenho preditivo.

1.3 Objetivos

Este trabalho tem como objetivo principal investigar a eficácia de algoritmos de aprendizado de máquina na detecção de *spyware*, a partir de dados extraídos da memória RAM, com foco na otimização de atributos relevantes e na interpretabilidade dos modelos por meio da aplicação do método SHAP.

Para atingir este objetivo, são estabelecidos os seguintes objetivos secundários:

- Realizar um estudo sobre as técnicas de análise de memória aplicadas à segurança da informação;
- Investigar os principais algoritmos de aprendizado de máquina;
- Pré-processar e selecionar atributos relevantes a partir de um conjunto de dados de memória (*dataset* Obfuscated-MalMem2022);
- Implementar e treinar modelos de detecção de *spyware* utilizando diferentes algoritmos de ML;
- Avaliar o desempenho dos modelos quanto à acurácia, sensibilidade, especificidade e tempo de inferência;

1.4 Divisão da Monografia

Esta monografia está organizada em cinco capítulos. O Capítulo 1 apresenta a introdução ao tema, contextualizando o problema de pesquisa, sua relevância, os objetivos do trabalho e a hipótese formulada. O Capítulo 2 aborda os fundamentos teóricos necessários para o desenvolvimento da pesquisa, incluindo conceitos sobre *spywares*, análise de memória, algoritmos de aprendizado de máquina e técnicas de interpretabilidade, com ênfase no método SHAP. No Capítulo 3, descreve-se a metodologia adotada, contemplando as etapas de preparação dos dados, escolha e configuração dos algoritmos, aplicação de seleção de atributos e estratégias de avaliação. O Capítulo 4, por sua vez, apresenta os resultados obtidos a partir dos experimentos realizados, acompanhados de análises comparativas e interpretações com base nas métricas definidas. Por fim, o Capítulo 5 expõe as conclusões do estudo, destacando as contribuições alcançadas, limitações observadas e sugestões para trabalhos futuros.

2 Fundamentação Teórica

Este capítulo apresenta os fundamentos teóricos que embasam a proposta deste trabalho, abordando o funcionamento dos *spywares* e sua atuação furtiva, a importância da análise da memória RAM como fonte de evidências comportamentais, e os principais algoritmos de aprendizado de máquina utilizados. Também são discutidas técnicas de seleção de atributos e o método SHAP, essencial para a interpretabilidade dos modelos adotados na detecção de ameaças.

2.1 Malware

O termo *Malware* é uma abreviação de *malicious software*, um termo referente a qualquer *software* desenvolvido com o objetivo de comprometer a **segurança**, **privacidade** e/ou **funcionamento** de sistemas computacionais. Contendo inúmeras categorias, dentre elas, **vírus**, ***trojans***, ***worms***, ***ransonwares*** e *spywares*(BENSAOUD; KALITA; BENSAOUD, 2024), *Malware* também podem ser projetados para excluir arquivos ou até mesmo roubar dados sensíveis — ações estas que não foram autorizadas pelo usuário.

Sistemas Operacionais como **Windows**, **Android**, **Linux** e **MacOS** fazem atualizações com frequência semanal e/ou quinzenal contra vulnerabilidades críticas encontradas em seus sistemas. Por outro lado, os atacantes (ou criadores) desses programas maliciosos estão sempre à procura de novas vulnerabilidades, sendo regularmente noticiadas novas histórias de *softwares* maliciosos. A exemplos, em 2022, um ataque cibernético vindo da Rússia, por um grupo conhecido como **Killnet**, teve como alvo os serviços *web* dos estados do **Colorado**, **Alabama**, **Alaska**, **Delaware**, **Connecticut**, **Florida**, **Mississippi** e Kansas(BENSAOUD; KALITA; BENSAOUD, 2024).

Um dos maiores desafios para as organizações seria a detecção eficaz destes *malwares*, especialmente em razão da evolução constante das técnicas de ofuscação e também do crescimento no uso de ferramentas automatizadas para se criar novos ataques — exigindo soluções de defesa mais sofisticadas e adaptativas, tais como o uso de aprendizado de máquina para detecção comportamental(ASLAN; SAMET, 2020).

2.2 Spyware

Spyware é o termo que se refere à categoria de *malware* que tem como objetivo monitorar, registrar e transmitir informações sensíveis do usuário sem o seu consentimento e de forma oculta. Diferente de outras categorias de *malware* que se manifestam de forma

explícita, como por exemplo o *ransomware*, o *spyware* é projetado para operar silenciosamente — até mesmo por longos períodos — a fim de capturar dados como **credenciais**, **atividades online** e **comportamento do sistema**(EGELE et al., 2007).

O *spyware*, devido ao seu caráter dissimulado, representa uma ameaça significativa e de **difícil detecção**, tanto que, na maioria das vezes, somente análises profundas do sistema (especialmente de memória volátil) conseguem revelar indícios de sua presença — seja por meio de processos escondidos, manipulação não autorizada de recursos do sistema e/ou chamadas a bibliotecas dinâmicas DLL(CARRIER, 2021). Os tipos mais comuns da categoria são ***Adware***, ***Cookies and E-mail tracking***, ***Browser Hijackers*** e ***Spybots***(JONASSON; SIGHOLM, 2005).

2.2.1 Adware

Este tipo de *spyware* pode realizar diversas ações, desde o monitoramento até a exibição de anúncios enquanto você navega. Normalmente, vem embutido em outros *softwares* de publicidade, podendo ser inofensivo ao não realizar a coleta ou a transferência de informações. Há diversos debates sobre o *Adware* realmente ser um *spyware* ou não — inclusive *Adwares* são frequentemente utilizados como disfarce para outro tipo de *spyware*: os ***keyloggers***, funcionando como fachada para estes se infiltrar em(JONASSON; SIGHOLM, 2005).

2.2.2 *Cookies* e rastreamento de E-mail

Essa variante de *spyware* também pode ser considerada uma forma passiva de *spyware* — normalmente não contém códigos e utiliza funções pré-existentes no navegador ou cliente de e-mail. Os *Cookies* armazenam informações do usuário em um servidor, permitindo que provedores de anúncio rastreiem seu comportamento em diversos sites(JONASSON; SIGHOLM, 2005).

2.2.3 Hijackers

Os ***Hijackers*** são programas que modificam o comportamento do navegador sem o consentimento prévio do usuário. Alguns são ativados ao acessar sites ou clicar em botões, alterando a página inicial para exibir anúncios. Em casos mais graves podem instalar objetos auxiliares de navegador, que registram todas as atividades do usuário incluindo as *Uniform Resource Locators (URL)* visitadas e as pesquisas feitas e transmite essa informação para terceiros(JONASSON; SIGHOLM, 2005)

2.2.4 Spybots

Spybots são uma das formas mais conhecidas de *spyware*, pois monitoram o comportamento do usuário e transmitem o dado para partes terceiras. Diferentemente de ***keyloggers***, os ***spybots*** têm inteligência para escolher as informações que vão capturar, registrando dados de formulários, listas de contatos, endereços visitados ou arquivos do computador.

2.3 Desafios na Detecção de Spywares

A detecção de *spywares* impõe uma série de desafios técnicos, especialmente pelo uso de técnicas avançadas de camuflagem, como a polimorfia, a metamorfose de código e o uso de comportamentos legítimos como disfarce. Muitos *spywares*, por exemplo, utilizam de *Application Programming Interface (API)* comuns do sistema operacional para executar suas tarefas — o que dificulta sua diferenciação de processos legítimos(JONASSON; SIGHOLM, 2005).

Com o aumento da incidência de *spywares*, diversas soluções comerciais foram desenvolvidas com o objetivo de identificar e remover essas ameaças. A maioria dessas ferramentas adota uma abordagem baseada em assinaturas, similar à utilizada por antivírus tradicionais, na qual o código binário de arquivos suspeitos é comparado a uma base de dados previamente catalogada. No entanto, esse método apresenta limitações significativas, sobretudo frente ao comportamento discreto dos *spywares*, que operam com mínima interferência no sistema, consomem poucos recursos e evitam ações que possam ser facilmente detectadas por soluções baseadas em padrões estáticos. Tais características dificultam a detecção eficaz por métodos convencionais, exigindo o desenvolvimento de técnicas mais dinâmicas e inteligentes para mitigar as ameaças (ABUALHAJ et al., 2024).

Além das características de *spywares* explicitadas anteriormente, a abordagem baseada em assinaturas apresenta uma limitação crítica sobre a necessidade constante de atualização das bases de dados, bem como a incapacidade de detectar novas variantes de *spyware* ou códigos maliciosos ofuscados (JONASSON; SIGHOLM, 2005).

Diante desse cenário, torna-se essencial o emprego de técnicas que considerem aspectos comportamentais e contextuais da execução dos processos no sistema, permitindo a identificação de ameaças mesmo quando disfarçadas ou desconhecidas. Nesse contexto, a análise da memória RAM juntammente com a utilização de AI se destaca como uma alternativa eficaz, uma vez que permite observar em tempo real o comportamento dos processos em execução, contribuindo significativamente para a detecção de atividades maliciosas não capturadas por métodos tradicionais (LEONEL; MOLINOS; MIANI, 2024).

2.4 Machine Learning e Inteligência Artificial na Segurança

Nos últimos anos, técnicas de *Artificial Intelligence (AI)* e *Machine Learning (ML)* têm transformado a forma como ameaças digitais são enfrentadas, especialmente diante da crescente complexidade e sofisticação de ataques, como os provocados por *spywares* (BENSAOUD; KALITA; BENSAOUD, 2024; LEONEL; MOLINOS; MIANI, 2024).

Esses agentes maliciosos operam de maneira discreta, evitando os mecanismos tradicionais de detecção, baseados em assinaturas e regras estáticas. Nesse cenário, a AI se destaca por sua capacidade de realizar análises comportamentais profundas e adaptativas, detectando padrões anômalos mesmo quando não há registros prévios da ameaça em bases de dados. Algoritmos como Decision Tree, Naive Bayes, Support Vector Machine (SVM) e Redes Neurais vêm sendo aplicados com sucesso na identificação de atividades maliciosas, atingindo acurácia próxima a 99% em modelos baseados em dados extraídos da memória RAM (LEONEL; MOLINOS; MIANI, 2024).

Além da eficácia preditiva, outro diferencial importante do uso de AI em segurança é a possibilidade de interpretabilidade dos modelos, especialmente por meio de técnicas como SHAP — que permitem compreender quais atributos foram mais relevantes para a decisão do algoritmo, tornando os sistemas mais transparentes e auditáveis (LEONEL; MOLINOS; MIANI, 2024).

A capacidade dos modelos de ML de aprender com novos dados, adaptar-se a variações de comportamento e identificar códigos ofuscados ou variantes inéditas reforça sua superioridade frente a abordagens tradicionais, conforme apontado por (ABUALHAJ et al., 2024), as quais mostram falhas de até 80% na detecção de *keyloggers* por antivírus convencionais.

Em suma, a integração da AI nos sistemas de defesa cibernética amplia a capacidade de proteção, oferecendo soluções proativas, escaláveis e com alto grau de precisão frente ao avanço constante das ameaças digitais.

2.5 Importância da Seleção de Atributos no Treinamento dos Modelos de IA

A seleção de atributos (*feature selection*) é uma etapa fundamental no desenvolvimento de modelos de aprendizado de máquina, especialmente em aplicações de segurança cibernética. Esse processo visa reduzir a dimensionalidade do conjunto de dados, mantendo apenas as variáveis mais relevantes para a classificação. Com isso, além de melhorar a eficiência computacional, reduz-se o risco de sobreajuste (*overfitting*) e aumenta-se a interpretabilidade do modelo (GUYON; ELISSEEFF, 2003).

A literatura destaca três abordagens principais para seleção de atributos: *filter methods*, *wrapper methods* e *embedded methods* (CHANDRASHEKAR; SAHIN, 2014). Métodos de filtragem utilizam métricas estatísticas para avaliar a relevância dos atributos antes da modelagem, enquanto métodos baseados em *wrapper* aplicam algoritmos preditivos para validar a importância das variáveis. Já os métodos *embedded* integram a seleção de atributos diretamente ao processo de aprendizagem do modelo, como ocorre com Árvores de Decisão e Redes Neurais.

Conforme evidenciado por (LEONEL; MOLINOS; MIANI, 2024), o uso de algoritmos como Árvore de Decisão, aliado a técnicas como SHAP, permite identificar os atributos com maior impacto na decisão do modelo, promovendo modelos mais enxutos, rápidos e transparentes. A seleção correta dos atributos reduz o risco de sobreajuste e aumenta a interpretabilidade das decisões. Além disso, modelos que analisam dados de memória precisam identificar padrões como o número de processos ativos, chamadas à DLL e uso de recursos do sistema, garantindo que apenas os atributos essenciais sejam considerados na predição.

Estudos recentes demonstram que a correta seleção de atributos pode não apenas otimizar o tempo de inferência dos modelos, mas também aumentar a robustez contra ataques evasivos que buscam ocultar atividades maliciosas (MOUSTAFA; CREECH, 2019).Isso reforça a necessidade de estratégias eficazes para determinar quais atributos devem ser mantidos ou descartados, promovendo modelos mais rápidos, confiáveis e compreensíveis no enfrentamento de ameaças cibernéticas. No contexto específico da detecção de *spywares*, a escolha de atributos torna-se ainda mais relevante, pois muitos sinais de presença maliciosa são sutis e podem estar mascarados entre centenas de variáveis irrelevantes.

2.6 Trabalhos Correlatos

Esta seção apresenta estudos relevantes que fundamentam a pesquisa. Para cada estudo apresentado, há uma breve explicação do que foi feito, o modo, e os resultados que foram obtidos — visando comprovar sua relevância para o presente trabalho.

2.6.1 Detecção Aprimorada de *Spyware* Utilizando Árvores de Decisão com Otimização de Hiperparâmetros

O estudo de Abualhaj et al. (2024) aborda o uso de **Árvores de Decisão** na detecção de *spyware*. O objetivo é aprimorar a precisão na identificação de ameaças através da **otimização de hiperparâmetros** garantindo um modelo robusto e adaptável. Os pesquisadores aplicaram técnicas de otimização de hiperparâmetros para ajustar o desempenho das árvores de decisão, utilizando o conjunto de dados CIC-Malmem-2022 que

contém informações detalhadas sobre *spyware*. O modelo foi implementado em **Python** e adotou diferentes configurações para melhorar a sua precisão. Os resultados mostraram uma acurácia de **99,97%** na detecção de *spyware*, demonstrando-se altamente eficaz. Os bons resultados denotam a grande importância da seleção de atributos e otimização dos modelos, validando o uso de ML para fortalecer a segurança cibernética.

2.6.2 Deteção de Spyware em Dispositivos Android com Aprendizado de Máquina

Com o objetivo de identificar aplicativos maliciosos que espionam informações sensíveis em ambiente Android, a pesquisa de Qabalin, Naser e Alkasassbeh (2022) desenvolveu um conjunto de dados com três classes principais: tráfego normal, tráfego gerado durante a instalação do *spyware* e tráfego com as operações de *spyware*. Utilizando o algoritmo de *Random Forest* para validação para duas abordagens, binária e multiclasse, o modelo construído alcançou **79%** de acurácia na classificação binária e **77%** na classificação multiclasse. Este trabalho reforça, pois, a importância da análise de tráfego de rede e da seleção de atributos para a melhora da precisão dos modelos.

2.6.3 Otimização da Seleção de Recursos por meio de Algoritmos de Aprendizado de Máquina e IA Explicável na Análise de Memória.

O estudo de Leonel, Molinos e Miani (2024) investiga a otimização da seleção de atributos na detecção de *ransonware*. Para isso, foram aplicados diferentes algoritmos de aprendizado de máquina para detectar *ransonware* por meio dos dados de memória e também exploradas técnicas de seleção de atributos como SHAP para melhor compreensão dos modelos. Posteriormente, foram comparados os modelos, e também analisado o impacto da seleção de atributos em sua precisão ao detectar *ransonware*. O referido trabalho demonstrou que a otimização dos atributos melhora a precisão e possibilita criar modelos mais eficientes, com necessidade de menos atributos para uma boa acurácia.

2.6.4 Aprendizado de Máquina para Detecção de *Malware* Ofuscado

O estudo de (MEZINA; BURGET, 2022) investigou a detecção de *malware* ofuscado, utilizando aprendizado profundo. Sua proposta se baseia no uso de **Redes Neurais Convolucionais** com camadas dilatadas para identificar padrões ocultos, sendo testada em diversos conjuntos de dados — com diversas variantes de *malware* ofuscado — e com diferentes configurações de hiperparâmetros. Os resultados obtidos nessa abordagem mostraram que o modelo de aprendizado profundo, ou *Deep Learning*, alcançou taxa de detecção superior a **90%**, indicando que a técnica superou modelos tradicionais de detecção baseada em assinatura — destacando a necessidade de estratégias adaptativas para

ameaças que utilizam técnicas evasivas — e que redes neurais profundas conseguem lidar com as mudanças de comportamento do *malware*.

Para sintetizar os principais pontos analisados nesta seção, a Tabela 1 foi elaborada, permitindo uma comparação direta entre as diferentes abordagens e o presente trabalho.

Trabalho	Múltiplos Modelos	Tempo de Inferência	Otimização de Features
(ABUALHAJ et al., 2024)	×	×	✓
(QABALIN; NASER; ALKASASSBEH, 2022)	×	×	×
(LEONEL; MOLINOS; MIANI, 2024)	✓	×	✓
(MEZINA; BURGET, 2022)	×	×	×
Presente trabalho	✓	✓	✓

Tabela 1 – Comparação entre trabalhos correlatos e o presente estudo

Fonte: Autor

2.6.5 Limitações e Considerações sobre a Comparação com Trabalhos Anteriores

Este trabalho foi fortemente baseado na metodologia proposta por Leonel, Molinos e Miani (2024), que utilizou algoritmos de aprendizado de máquina e técnicas de seleção de atributos, como o SHAP, para detectar *ransomwares* por meio da análise de memória do dataset CIC-MalMem-2022. A adaptação desta abordagem para a detecção de *spywares* se deu pelo reconhecimento de que, embora ambos sejam tipos de *malware*, apresentam características de atuação distintas que podem impactar diretamente o comportamento dos modelos preditivos.

Inicialmente, entre os objetivos deste trabalho, estava prevista uma comparação direta dos resultados obtidos com aqueles reportados por Lucas Leonel, a fim de averiguar a possibilidade de um eventual **enviesamento no dataset CIC-MalMem-2022**. A hipótese era que, ao replicar a metodologia para outra classe de *malware* — o *spyware*, em vez de *ransomware* — seria possível verificar se o conjunto de dados apresentava características que favorecessem de forma desproporcional determinadas categorias de ataque.

No entanto, essa comparação não pôde ser realizada neste trabalho devido à **limitação temporal** imposta pelo cronograma da pesquisa, assim como pela necessidade de priorizar a implementação e avaliação dos modelos para a nova categoria de *spyware*. Embora os resultados obtidos neste estudo tenham sido satisfatórios, a análise comparativa com o trabalho de Leonel permanece uma perspectiva valiosa para investigações futuras.

A realização dessa análise poderia fornecer insights importantes sobre a **generalização e robustez** do dataset CIC-MalMem-2022 frente a diferentes classes de *malware*, bem como contribuir para a avaliação de potenciais vieses que possam impactar a qualidade dos modelos treinados com esse conjunto de dados.

3 Metodologia

Torna-se relevante destacar que o método prático de experimentação adotado neste trabalho segue integralmente o mesmo protocolo descrito por (LEONEL; MOLINOS; MIANI, 2024). Isso inclui os algoritmos de aprendizado de máquina, as configurações dos modelos e o uso do mesmo *dataset (CIC-MalMem-2022)*. A única diferença reside no tipo de amostra analisada: enquanto LEONEL; MOLINOS; MIANI focam em instâncias de *ransomware*, este estudo concentra-se especificamente em dados classificados como *spyware*.

A adoção dessa abordagem comum visa estabelecer uma base comparativa robusta entre os diferentes tipos de *malware* presentes no conjunto de dados. Trata-se de uma estratégia intencional para possibilitar uma análise mais abrangente da qualidade do *dataset* por meio de métodos de inteligência artificial explicável (XAI). Assim, assegura-se a consistência metodológica e minimizam-se possíveis vieses analíticos decorrentes de variações no processo experimental.

Este capítulo se destina a detalhar o método de pesquisa adotado para este trabalho, sendo as principais contribuições: (i) descrever a sequência de passos realizados na construção da proposta; e (ii) definição do escopo de trabalho.

A figura 1 ilustra de forma simplificada as etapas da metodologia proposta para o trabalho

3.1 Percepção da problemática

A segurança digital é um campo que demanda soluções cada vez mais inteligentes, autônomas e adaptativas. Dentro desse cenário, a detecção de *spywares* representa um desafio crescente, uma vez que se trata de uma ameaça silenciosa, cuja presença dificilmente é identificada por abordagens tradicionais baseadas em assinaturas ou em comportamentos predefinidos. Diferentemente de malwares destrutivos, como os *ransomwares*, os *spywares* não interrompem o funcionamento do sistema, o que contribui para sua baixa perceptibilidade e prolongada permanência no ambiente comprometido.

Diante dessa dificuldade, este trabalho propõe a aplicação de técnicas de aprendizado de máquina para identificar comportamentos suspeitos por meio da análise da memória volátil do sistema. Dados extraídos da RAM permitem capturar o estado real do sistema em execução, oferecendo uma visão precisa e detalhada da atividade dos processos ativos.

Essa abordagem possibilita a extração de características comportamentais úteis à

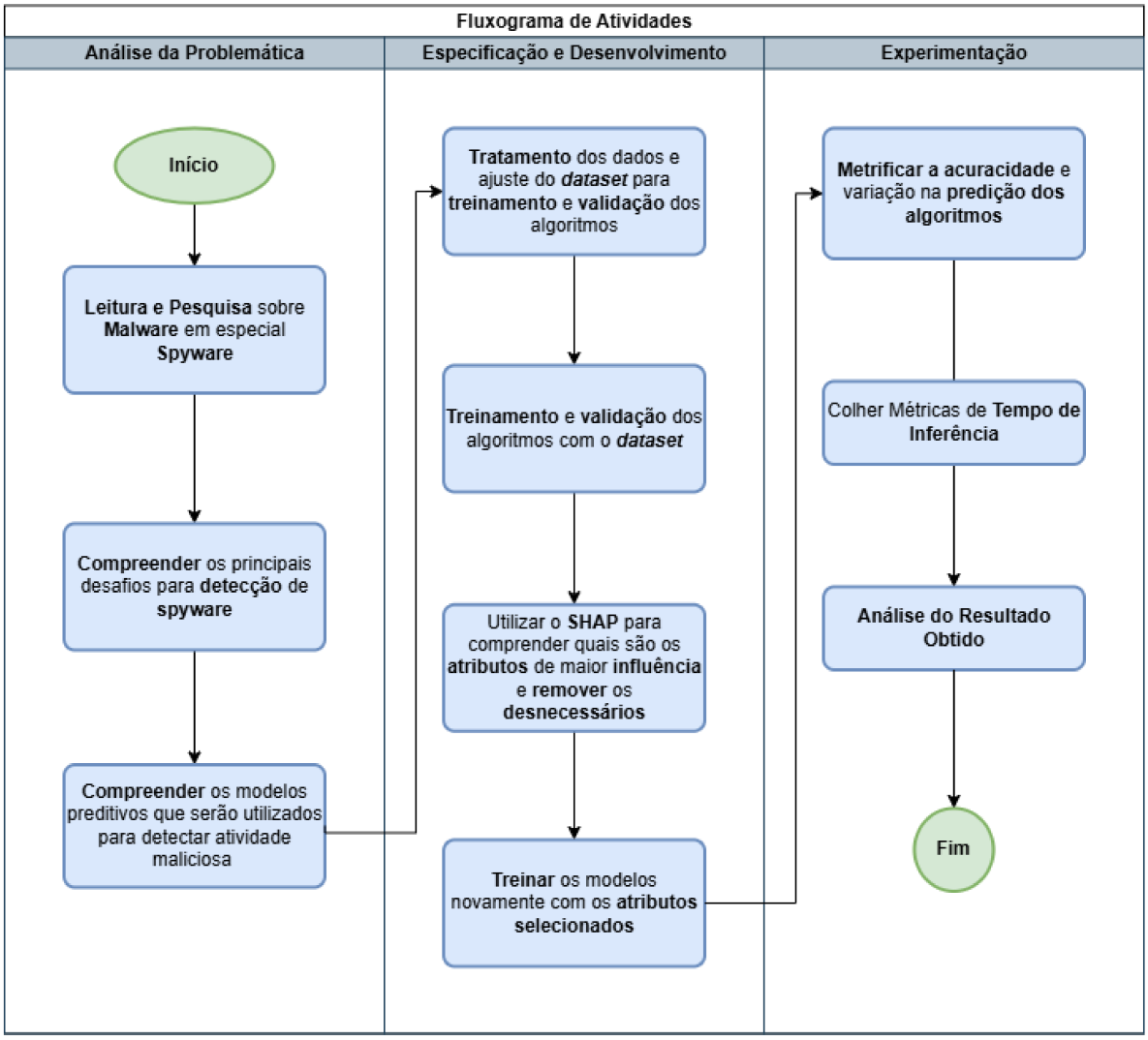

Figura 1 – Metodologia Proposta

Fonte: Autor

classificação automática de instâncias maliciosas, mesmo na ausência de sinais evidentes de ataque.

3.1.1 Desafios na Detecção de Spyware

A revisão da literatura foi essencial para compreender as abordagens existentes para detecção de *malware* por análise de memória. Estudos como o de (LEONEL; MOLINOS; MIANI, 2024) e (ABUALHAJ et al., 2024) evidenciaram o potencial da integração entre algoritmos de ML e técnicas de inteligência artificial para detectar *ransomwares* e *spywares* com alta acurácia.

Fato é que a literatura encontra-se inundada de trabalhos que exploram a detecção de *malwares* de um modo geral. No entanto, estudos que tratam especificamente de *spyware* são menos frequentes, revelando uma lacuna no uso de evidências comportamen-

tais em memória para esse tipo de ameaça.

3.2 Especificação e Desenvolvimento

Nesta seção do método, descrevem-se as etapas práticas adotadas para implementação da proposta deste trabalho, abrangendo desde a seleção e preparação do conjunto de dados até a aplicação dos algoritmos de aprendizado de máquina e das técnicas de interpretabilidade.

3.3 Preparação dos Dados

O conjunto de dados utilizado neste trabalho é o Cybersecurity (2022), desenvolvido pelo *Canadian Institute for Cybersecurity (CIC)* — contendo amostras de memória extraídas de sistemas sob diferentes tipos de infecções maliciosas, incluindo *ransomware*, *trojan*, *spyware* e amostras *benignas*. Este *dataset* foi escolhido pela sua vasta base, composta por **58.596 registros** e **55 atributos**, distribuídos entre características relacionadas a processos, módulos carregados, acessos a arquivos, conexões de rede, DLL e serviços. Os dados foram organizados em **20 categorias**, e cada uma representa uma família específica de *malware*, como *trojan*, *ransonware* e *spyware*.

3.3.1 Pré-processamento do *Dataset*

Para este estudo, foram selecionadas exclusivamente as amostras classificadas como ***spyware*** e ***benignas***, compondo um conjunto binário de classificação. O pré-processamento inclui:

- Tratamento de valores ausentes e *outliers*;
- Normalização dos dados com *StandardScaler*.
- Remoção de colunas não numéricas ou irrelevantes (ex: identificadores);
- Filtragem de registros por categoria (*spyware* e *benignos*);

As atividades de leitura, processamento e transformação dos dados do *dataset* foram conduzidas com o auxílio da biblioteca **Pandas**, amplamente utilizada para manipulação de dados tabulares.

3.4 Seleção dos Algoritmos

A escolha de algoritmos para compor o modelo de referência, também conhecido como *baseline*, é uma etapa essencial no desenvolvimento de soluções baseadas em aprendizado de máquina. O *baseline* funciona como ponto de partida para a avaliação de técnicas mais avançadas, permitindo comparar seu desempenho com abordagens tradicionais e, assim, verificar se os ganhos obtidos são efetivamente relevantes.

Neste trabalho, foram selecionados quatro algoritmos clássicos, cada um representando uma classe distinta dentro do campo do aprendizado de máquina. A escolha visa garantir diversidade metodológica, facilitando a análise comparativa entre diferentes paradigmas de classificação. Os modelos escolhidos foram:

- **Rede Neural Artificial** — pertencente à classe de *Aprendizado Profundo*, destaca-se pela capacidade de capturar relações complexas entre os dados. É altamente flexível, mas possui custo computacional elevado e exige cuidado com *overfitting*.
- **Árvore de Decisão** — modelo baseado em *regras*, conhecido pela fácil interpretação e bom desempenho com dados tabulares. Pode, entretanto, sofrer de sobreajuste se não for regulado corretamente.
- **Naive Bayes** — classificador de natureza *estatística e probabilística*, é rápido, eficaz com conjuntos de dados pequenos e de alta dimensionalidade. Sua principal limitação é a suposição de independência entre as variáveis.
- **Support Vector Machine (SVM)** — pertence ao segmento de *classificadores lineares com margem máxima*. Apresenta alta precisão, especialmente quando os dados são bem separados, mas pode ser lento em bases de grande volume e sensível a ajustes de parâmetros.

Essa diversidade algorítmica possibilita uma avaliação abrangente do problema de detecção de *spywares*, comparando modelos simples e interpretáveis com abordagens mais sofisticadas e robustas.

3.5 Treinamento

Os modelos foram treinados utilizando a divisão ***Holdout***, com divisão de **80%** dos dados para treinamento e **20%** para teste. Além disso, foi aplicada a técnica de **validação cruzada estratificada** com $k = 5$ (*5-fold cross-validation*).

O conjunto de dados foi dividido em cinco subconjuntos (**folds**), e, a cada iteração, quatro **folds** foram utilizados para o treinamento e um para a validação. Esse processo

foi repetido cinco vezes, garantindo que todas as amostras fossem usadas tanto no treinamento quanto na validação, promovendo uma avaliação mais estável e confiável dos modelos. A média das métricas obtidas em cada rodada foi utilizada como resultado final, permitindo observar a consistência do desempenho entre diferentes partições do conjunto de dados.

A implementação dos modelos de aprendizado de máquina foi realizada com as bibliotecas **Scikit-learn** e **Keras**. A Scikit-learn foi empregada para os classificadores *Naive Bayes* (GaussianNB), Árvore de Decisão(DT) e Support Vector Machine (SVM), enquanto a biblioteca Keras (com *framework* **TensorFlow**) foi utilizada para a construção da Rede Neural.

3.5.1 Parâmetros dos Modelos

Todos os modelos foram submetidos à mesma divisão de dados, com validação cruzada estratificada (**k-fold** com $k = 5$) e normalização dos atributos via ***StandardScaler***. Essa padronização visou garantir consistência na comparação entre os algoritmos.

- **Rede Neural Artificial**: A rede foi construída com cinco camadas, sendo quatro ocultas e uma de saída, utilizando a seguinte configuração:
  - Camada 1: **Dense** com **15** neurônios e função de ativação **ReLU**;
  - Camada 2: **Dropout** com taxa de **0.1** para prevenir *overfitting*;
  - Camada 3: **Dense** com **15** neurônios e ativação **ReLU**;
  - Camada 4: **Dense** com **8** neurônios e ativação **ReLU**;
  - Camada de saída: **Dense** com **1** neurônio e ativação **Sigmoid**.

  A rede foi treinada por **100** épocas, com **batch_size = 10**, utilizando o otimizador **Adam** e a função de perda **binary_crossentropy**, adequada para problemas de classificação binária.

- **Árvore de Decisão**: O modelo foi treinado com os parâmetros padrão da Scikit-learn, utilizando o critério **gini** para cálculo de impureza. Não foi definida profundidade máxima ou número mínimo de amostras por folha, permitindo que a árvore crescesse livremente para observar sua capacidade de ajuste natural ao conjunto.

- **Naive Bayes**: Utilizou o classificador **GaussianNB**, assumindo distribuição normal dos atributos. Por ser um algoritmo de baixa complexidade, foi mantido com os parâmetros padrão, focando na eficiência e simplicidade do modelo.

- **Support Vector Machine (SVM)**: Foi configurado com **kernel='linear'** e parâmetro de regularização **C=0.1**.

Para medir o tempo de inferência, foram implementadas duas abordagens:

- **Execução individual**: utilizando **time.perf_counter()** para medir o tempo de inferência para cada amostra benigna ou de *spyware*, fazendo a inferência de uma instância por vez.
- **Execução em bloco**: utilizando **timeit** com 100 execuções consecutivas do conjunto de amostras, de modo a reduzir variações e obter tempo médio por amostra.

Todos os modelos foram testados em cenários idênticos, com as mesmas amostras benignas e de *spyware*, técnicas de validação e métricas de avaliação. Essa uniformidade metodológica permite comparar de forma justa o desempenho entre os algoritmos, tanto em termos de acurácia quanto em tempo de inferência. Torna-se importante ressaltar que o método de treinamento utilizado nesta proposta é o detalhado em (LEONEL; MOLINOS; MIANI, 2024).

3.5.2 Baseline

O *baseline* foi definido com o objetivo de estabelecer um ponto de partida para comparação entre os modelos. Para cada um dos quatro algoritmos selecionados foram gerados gráficos de curvas de aprendizado e matriz de confusão.

As curvas de aprendizado foram construídas com base na evolução da **acurácia** e da **função de perda** (*loss*) durante o treinamento, foi utilizada a técnica de curvas de aprendizado. Esta abordagem permite analisar a evolução da acurácia do modelo conforme a quantidade de dados de treinamento é progressivamente aumentada.

As curvas foram geradas utilizando a função learning_curve da biblioteca *scikit-learn*. Esta função realiza, para cada fração especificada do conjunto de dados de treinamento, uma validação cruzada (*k-fold cross-validation*), sendo que, neste experimento, foi adotado o valor de k = 10.

Para cada fração de dados (variando de 10% até 100% do conjunto de treinamento), o seguinte processo foi realizado:

- O subconjunto correspondente é separado do conjunto total de treinamento.
- Este subconjunto é dividido em 10 partições (folds) estratificadas.
- O modelo é treinado 10 vezes, cada uma utilizando 9 folds para treinamento e 1 fold para validação.
- A acurácia média e o desvio padrão da acurácia de treinamento e validação são calculados e armazenados.

Este procedimento foi repetido para diferentes tamanhos de subconjuntos, conforme configurado pelo parâmetro train_sizes=np.linspace(0.1, 1.0, 10), totalizando 10 pontos distintos na curva de aprendizado para cada modelo avaliado. Essas curvas permitem identificar três possíveis comportamentos dos modelos:

- **Overfitting**: quando há alto desempenho no conjunto de treino, mas baixo desempenho no conjunto de teste;
- **Underfitting**: quando o modelo apresenta baixo desempenho em ambos os conjuntos;
- **Ajuste adequado**: quando o modelo mantém bom desempenho e estabilidade em treino e teste.

A matriz de confusão foi gerada com base no conjunto de teste, correspondente a **20%** do total de dados, ou seja, aproximadamente **2000** amostras por **classe**. Isso explica o número de amostras observadas na matriz, com valores próximos de **2000** tanto para amostras **benignas** quanto para amostras de ***spyware*** e são compostas por quatro elementos:

- **TP (True Positives)**: casos de *spyware* corretamente classificados como *spyware*;
- **TN (True Negatives)**: casos *benignos* corretamente classificados como *benignos*;
- **FP (False Positives)**: casos *benignos* incorretamente classificados como *spyware*;
- **FN (False Negatives)**: casos de *spyware* incorretamente classificados como *benignos*.

A partir desses valores, foram calculadas as principais métricas de avaliação:

- **Acurácia**: $\text{ACC} = \frac{TP+TN}{TP+TN+FP+FN}$
- **Taxa de Verdadeiros Positivos (TPR)**: $\text{TPR} = \frac{TP}{TP+FN}$
- **Taxa de Falsos Positivos (FPR)**: $\text{FPR} = \frac{FP}{FP+TN}$

Essas métricas, juntamente com as curvas de aprendizado, possibilitam avaliar o desempenho geral dos modelos e sua capacidade de detectar corretamente as ameaças (**TPR**) ao mesmo tempo em que minimizam alarmes falsos (**FPR**). Nesta etapa, as bibliotecas **Matplotlib**, **Seaborn** e **Plotly** foram utilizadas para geração de visualizações, gráficos de análise exploratória e curvas de aprendizado.

3.6 Otimização de *Features*

A etapa de seleção de atributos é crucial no desenvolvimento de modelos de aprendizado de máquina, principalmente em domínios como segurança cibernética, onde há uma grande quantidade de variáveis e muitas podem ser irrelevantes ou redundantes. No contexto da detecção de *spywares*, selecionar os atributos mais representativos significa reduzir ruídos nos dados, evitar sobreajuste e melhorar tanto a acurácia quanto o tempo de inferência dos modelos.

Nesta etapa, o principal objetivo foi reduzir a quantidade de atributos utilizados na construção dos modelos, buscando preservar o desempenho preditivo. Para isso, foi adotada uma abordagem baseada em interpretabilidade, utilizando as técnicas de *Feature Importance* e SHAP na Árvore de Decisão. Essa abordagem permite analisar a contribuição individual de cada atributo na decisão dos modelos, promovendo maior transparência e compreensão do comportamento dos algoritmos. A partir da análise dos valores SHAP, foi possível identificar quais atributos apresentaram maior influência nas predições, auxiliando na seleção das variáveis mais relevantes para a tarefa de detecção de *spyware*.

3.6.1 Feature Importance e SHAP

A utilização da técnica *Feature Importance* no classificador de Árvore de Decisão se justifica por sua simplicidade, interpretabilidade e compatibilidade com dados tabulares, como os do **CIC-Malmem-2022**. Essa técnica classifica os atributos com base em sua capacidade de reduzir a impureza em cada divisão da árvore, permitindo identificar aqueles com maior poder discriminativo.

Além disso, foi empregada a abordagem SHAP, uma técnica de interpretabilidade baseada em teoria dos jogos, que permite explicar o impacto de cada atributo na predição de um modelo. Utilizar o SHAP possibilita não apenas selecionar os melhores atributos, mas também entender seu comportamento ao influenciar decisões.

A partir dessas análises, foi possível reduzir o número de atributos sem comprometer a capacidade de classificação dos modelos, otimizando o desempenho e melhorando a interpretabilidade dos resultados.

3.7 Treinamento de Novos Modelos após Otimização do Conjunto de Features

Inicialmente, foram utilizados **55 atributos originais** do dataset **CIC-MalMem-2022**, para todo o treinamento e validação inicial propostos pelo *baseline*. Após a otimização dos modelos removendo os atributos de menor impacto para os algoritmos, foram

treinados novos modelos a partir de um conjunto de atributos selecionados — com apenas **9 atributos** a fim de garantir mais desempenho e melhorar a interpretabilidade dos modelos gerados.

3.8 Tempo de Inferência dos Modelos

A fim de medir não somente a eficácia dos novos modelos — método conhecido como avaliação estática — também foi feita uma análise do tempo de inferência de cada um deles, em que foi possível comparar os percentuais estáticos de treinamento com a capacidade dinâmica do modelo em detectar instâncias maliciosas.

Para análise de inferência, foram utilizados dois cenários com diferentes quantidades de dados para validação. Neste caso, avaliar o tempo gasto pelo modelo treinado com relação à uma instância, classificando como benigno ou maligno.

Para esta etapa, foram utilizados as bibliotecas **timeit** para os tempos de inferências em bloco, a biblioteca time para cálculo da inferência individual e a biblioteca *Matplotlib* para gerar gráficos comparativos.

3.8.1 Tempo por instância individual

Para o cálculo de instância de maneira individual foi usado um laço de repetição com **100 exemplos**, somados os tempos de inferência para cada um deles e, depois, o valor foi dividido por **100**.

3.8.2 Tempo por Bloco

Para o cálculo do tempo de inferência por bloco, utilizou-se a função *timeit*, que permite executar um mesmo bloco de código múltiplas vezes. Essa repetição auxilia a reduzir a variância causada por oscilações pontuais no tempo de execução, proporcionando uma estimativa mais precisa.

3.8.3 Comparativo de tempo por instância individual e tempo de execução em bloco

Para fins de comparação e exemplificação da diferença entre ambas as abordagens de inferência, foi realizado um experimento comparativo utilizando gráficos de colunas. O objetivo foi visualizar o tempo necessário para que cada modelo classificasse os exemplos como infectados ou não, considerando dois cenários distintos: inferência individual e inferência em bloco.

Nesta etapa, foram utilizados dois subconjuntos de validação, cada um contendo **100 amostras benignas e 100 amostras de *spyware***. O tempo de inferência foi medido separadamente para cada abordagem: no modo individual, cada amostra foi processada uma a uma dentro de um laço de repetição; já no modo em bloco, os **100 exemplos** foram classificados de uma única vez. Essa análise permitiu avaliar não apenas a precisão dos modelos, mas também sua viabilidade operacional em cenários de uso real.

3.8.4 Comparativo de tempo de execução em bloco para diferentes tamanhos de bloco

Este comparativo teve como objetivo analisar a influência do tamanho dos subconjuntos de dados no tempo de execução dos modelos. Para isso, foram medidos os tempos de inferência em blocos de diferentes tamanhos: **100**, **200** e **300** exemplos de dados de memória.

Os resultados foram apresentados por meio de gráficos de colunas, permitindo visualizar como o desempenho dos modelos varia conforme a quantidade de instâncias processadas em lote. Essa análise é relevante para compreender a escalabilidade e eficiência dos algoritmos em cenários que exigem processamento contínuo ou ainda em grandes volumes de dados.

3.9 Validação e Resultado

Para validar o desempenho dos algoritmos aplicados à detecção de *spywares*, foi empregada uma abordagem quantitativa baseada em particionamento do conjunto de dados e avaliação por meio de métricas padrão de classificação.

Os modelos foram avaliados com base em métricas como acurácia, precisão, sensibilidade (recall) e F1-score, permitindo uma análise abrangente do desempenho preditivo. Além das métricas quantitativas, aplicou-se a técnica SHAP para interpretar os resultados dos modelos e identificar quais atributos extraídos da memória tiveram maior influência nas decisões de classificação. Essa análise possibilitou compreender o comportamento dos algoritmos e avaliar sua aplicabilidade prática em cenários reais de segurança cibernética.

Adicionalmente, foi avaliado o tempo de inferência de cada modelo durante a detecção de amostras benignas e maliciosas, possibilitando uma análise comparativa do custo computacional associado à aplicação prática dos classificadores em ambientes de tempo real. Dessa forma, os resultados obtidos reforçam o potencial do uso combinado de aprendizado de máquina com técnicas explicáveis para a construção de sistemas de segurança mais eficientes, transparentes e escaláveis.

4 Experimentação

Esta seção expõe a análise do experimento proposto no capítulo anterior e os resultados obtidos. Em resumo, os modelos foram treinados usando o *dataset* original com todos os atributos a fim de estimar a performance desses algoritmos. Através da análise de dados e resultados obtidos no passo anterior, o *dataset* e os modelos foram otimizados com a remoção dos atributos não essenciais.

4.1 Classificação do *Baseline*

A base de dados, conta com **10020** resultados relacionados à *spyware* e divididos por tipo: **2000** (*180solutions*), **2410** (*Transponder*), **2000** (CWS), **1410** (TIBS), **2200** (*Gator*). O *dataset* também conta com **29298** exemplos de dados **benignos**. Foram utilizados todos os exemplos relacionados a *spyware* para este trabalho — e o *baseline* ficou organizado da seguinte maneira:

- Instalação das dependências no *kernel*
- Leitura e Tratamento do *dataset*
- Rede Neural(Treinamento, Validação e Avaliação)
- Árvore de Decisão(Treinamento, Validação e Avaliação)
- Naive Bayes(Treinamento, Validação e Avaliação)
- SVM(Treinamento, Validação e Avaliação)

4.2 Treinamento Inicial - *Baseline*

No treinamento inicial realizado no *baseline*, todos os modelos gerados obtiveram resultados bastante satisfatórios em termos de acurácia, em que os modelos já demonstram um grande potencial e robustez para detectar *spywares* por meio da análise de memória. Os resultados obtidos e as análises de resultados serão apresentados seguindo a estrutura proposta na seção 4.1, modelo a modelo.

4.2.1 Rede Neural

O modelo foi treinado por **100** épocas com *batchsize* de **10** amostras, e manteve o balanceamento entre as classes benigno e *spyware*. Os gráficos da Figura 2 apresentam as curvas de aprendizado do modelo.

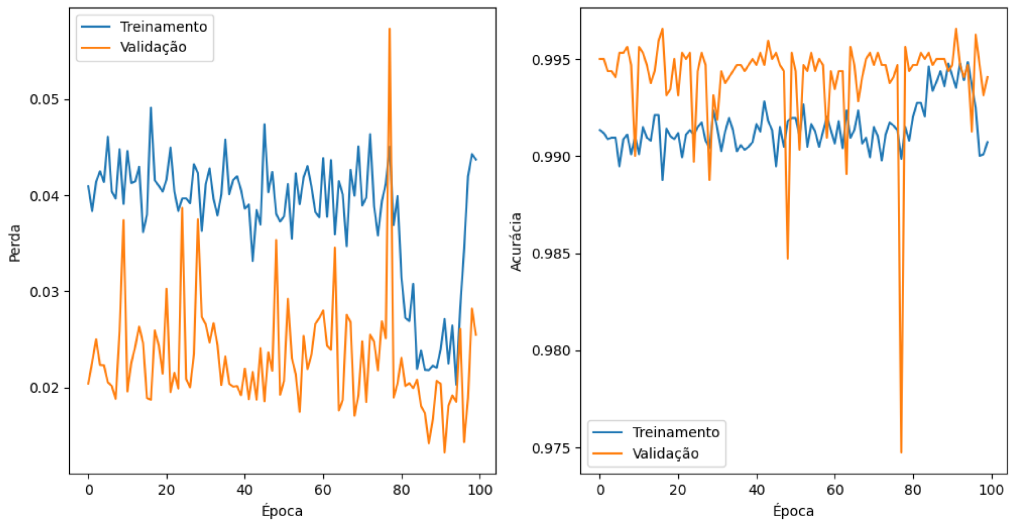

Figura 2 – Curvas de Aprendizado - Rede Neural

Fonte: Autor

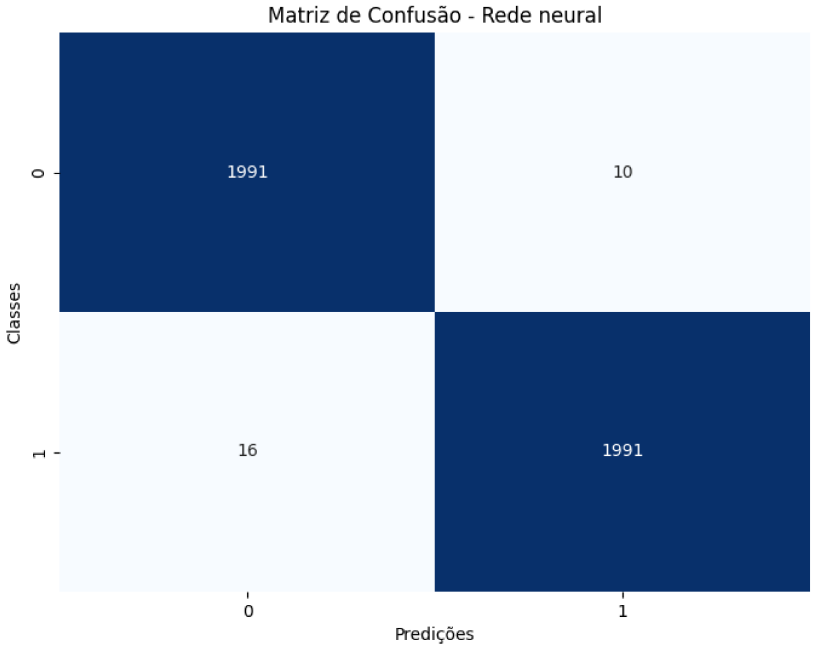

Figura 3 – Matriz de Confusão - Rede Neural

Fonte: Autor

Analisando as curvas de aprendizado da Figura 2, observa-se perdas pontuais ao longo do treinamento e validação com oscilações moderadas ao longo das épocas, mas sem tendência de divergência. Além disso, ao se deparar com os valores da coluna de perda, nota-se variações extremamente baixas, indicando, a priori, um processo de aprendizado estável. Ademais, a acurácia se mantém elevada, com valores superiores a **98,5%** em

ambas as fases de treinamento e validação, com pequenos picos de instabilidade — logo pode-se dizer que o modelo aprende a distinguir adequadamente entre amostras benignas e infectadas.

A matriz de confusão apresentada pelo modelo na Figura 3 reforça ainda mais o desempenho robusto, demonstrando em números a performance da Rede Neural. Foram observados **1991** verdadeiros positivos e **1991** negativos, com apenas **10** falsos positivos e **16** falsos negativos. Esses resultados indicam que o modelo foi capaz de identificar de forma correta a grande maioria das amostras, tanto benignas quanto amostras de *spyware*, mantendo o índice de erro muito baixo.

De maneira geral, o comportamento das curvas de perda e acurácia, aliados a matriz de confusão gerada, comprovam que a rede neural atingiu uma boa capacidade de generalização sem apresentar sinais de sobreajuste(*overfitting*) ou subajuste (*underfitting*).

4.2.2 Árvore de Decisão

O modelo de Árvore de Decisão apresentou o melhor desempenho em termos de acurácia dentre os modelos gerados no *baseline*. Já a curva de acurácia do modelo apresentou **100%** de assertividade durante o treinamento e **99,8%** durante a validação — ligeiramente inferior, mas ainda muito alta.

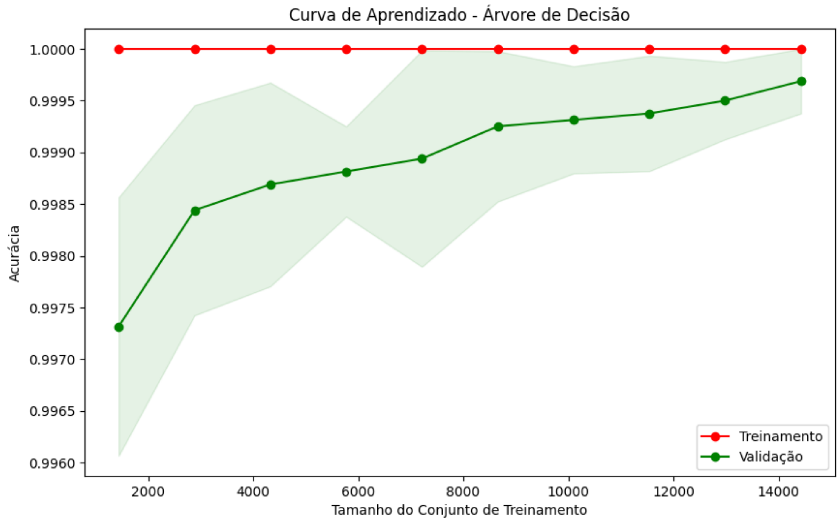

Figura 4 – Curva de Aprendizado - Acurácia - Árvore de Decisão

Fonte: Autor

O comportamento apresentado nas curvas de aprendizado de Acurácia(Figura 4) e Perda(Figura 5) sugere sinais de sobreajuste(*overfitting*), uma vez que esse tipo de algoritmo tende a memorizar os dados de treinamento. Porém, há uma convergência da validação conforme o volume de dados aumenta.

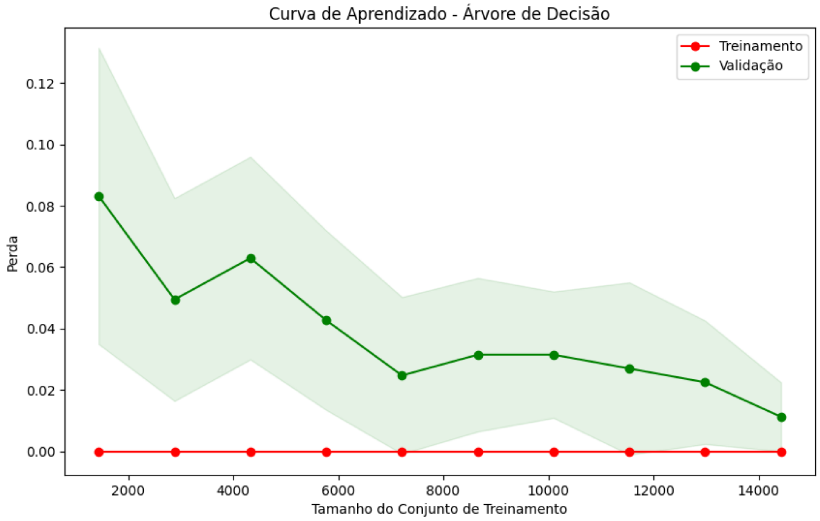

Figura 5 – Curva de Aprendizado - Perda - Árvore de Decisão

Fonte: Autor

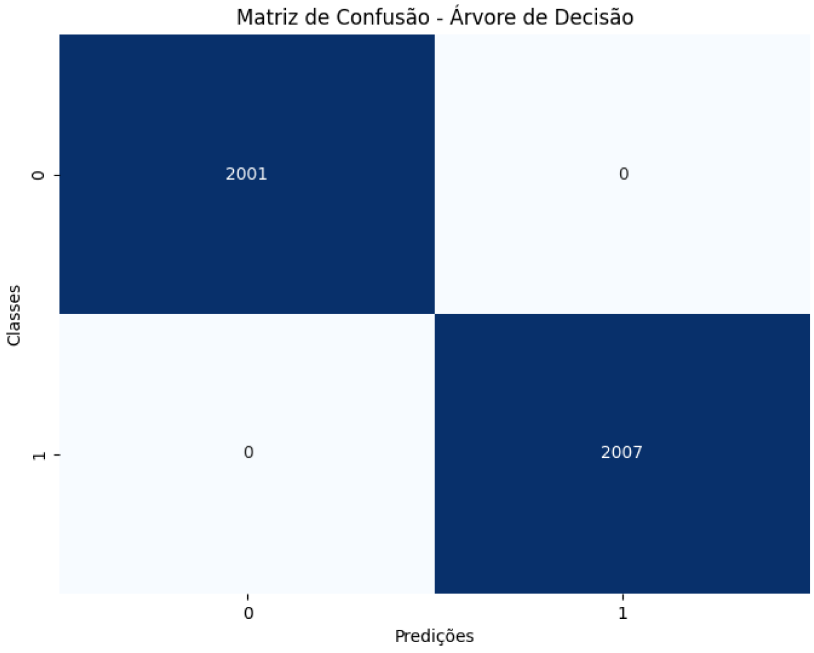

Figura 6 – Matriz de Confusão - Árvore de Decisão

Fonte: Autor

A matriz de confusão do modelo apresentada na Figura 6 evidencia que o modelo acertou todas as amostras de teste, sem registros de falsos positivos ou falsos negativos. Em resumo, o modelo apresentou altíssima acurácia, mas com indícios de *overfitting*, o que pode comprometer sua capacidade de generalização.

4.2.3 Naive Bayes

O modelo **Naive Bayes** também apresentou alta acurácia para a detecção de *spyware*. Conforme mostrado na Figura 7, o modelo apresentou uma acurácia próxima de **99,1%**, acompanhado de uma uma boa estabilidade tanto no treinamento quanto na validação. A baixa variação entre as curvas indica uma boa capacidade de generalização.

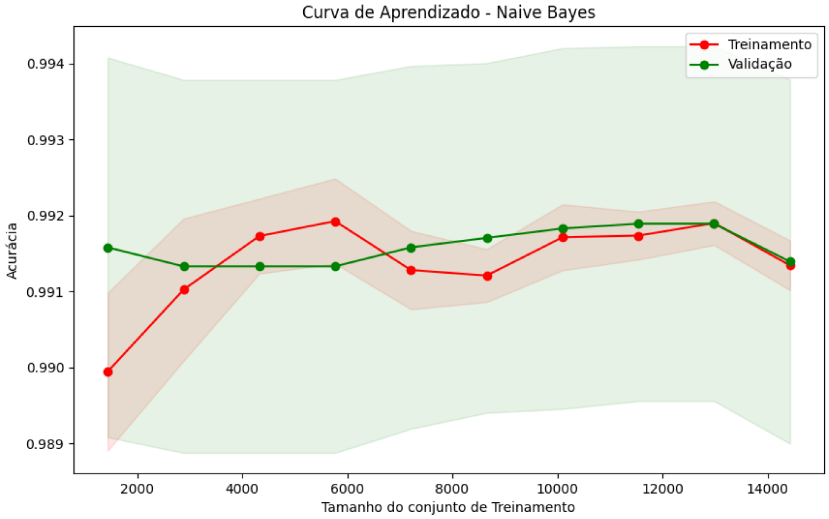

Figura 7 – Curva de Aprendizado - Acurácia - Naive Bayes

Fonte: Autor

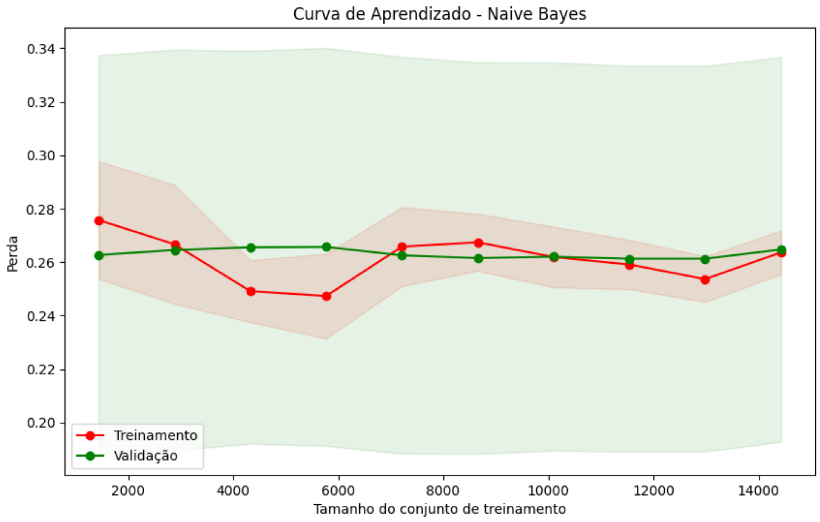

Figura 8 – Curva de Aprendizado - Perda - Naive Bayes

Fonte: Autor

A curva de perda, exibida na Figura 8, mostra que tanto a perda durante o treinamento como na validação permanceram estáveis, demonstando consistência do modelo.

Outro ponto que chama atenção no gráfico é a faixa verde ao fundo que representa a variação desses resultados, porém, mesmo com uma faixa de variação alta, a escala do gráfico permanece baixa, sendo assim não caracteriza uma instabilidade.

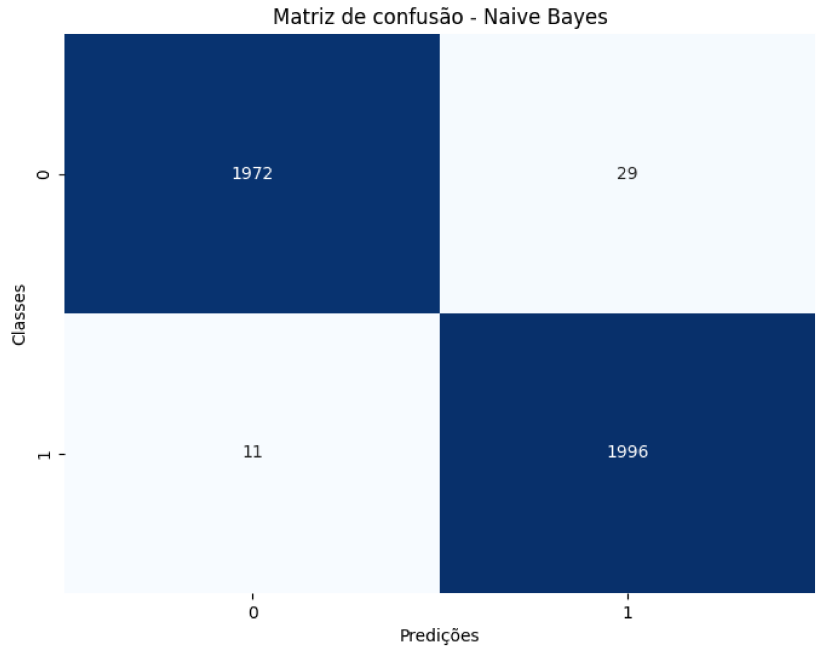

Figura 9 – Matriz de Confusão - Naive Bayes

Fonte: Autor

A matriz de confusão também apresentou uma performance exemplar, classificando a esmagadora maioria das amostras — apresentando **1972** verdadeiros negativos e **1996** verdadeiros positivos. Porém, ocorreram **29** falsos positivos e **11** falsos negativos, sugerindo que o modelo teve dificuldades para classificar algumas amostras de maneira correta. Apesar da (pequena) taxa de erro, o modelo demonstrou um bom desempenho geral, evidenciado pela estabilidade entre curvas de treino e validação, e baixo índice de perda.

4.2.4 SVM

O modelo SVM apresentou uma ótima capacidade de detecção de *spyware*. De acordo com a Figura 10, sua curva de acurácia obteve um índice muito próximo de **100%** durante o treinamento e um pouco menor ao longo da validação — mas também dentro dos **99%**, o que indica uma boa capacidade de ajuste. A distância entre as curvas é pequena e diminui conforme o aumento do conjunto de treinamento, o que sugere uma convergência e aprendizado adequados.

A Figura 2 exibe uma curva de perda, tanto em treinamento quanto em validação, descrescente e estável conforme o volume de amostras aumenta — confirmando que o

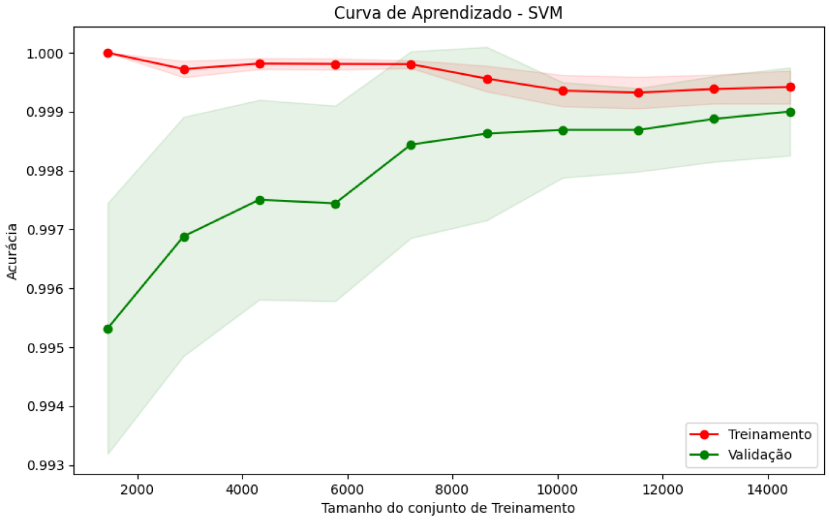

Figura 10 – Curva de Aprendizado - Acurácia - SVM

Fonte: Autor

modelo melhora sua performance à medida que mais dados são usados no treinamento.

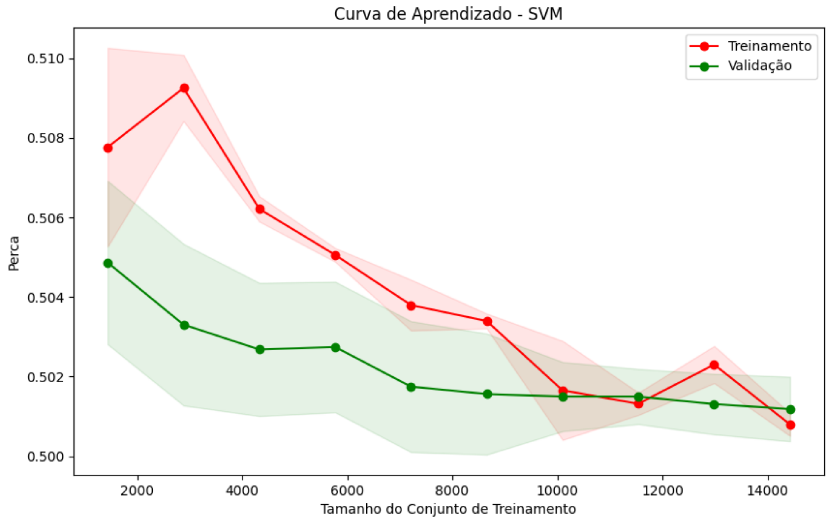

Figura 11 – Curva de Aprendizado - Perda - SVM

Fonte: Autor

Os resultados da matriz de confusão do modelo SVM, na Figura 12, confirmam ainda mais sua eficiência, com um desempenho execelente. O modelo obteve **1999** verdadeiros negativos e **2005** verdadeiros positivos, com apenas **2** falsos positivos e **2** falsos negativos. Tais números comprovam uma alta precisão do classificador tanto para exemplos de amostras benignas quanto infectadas.

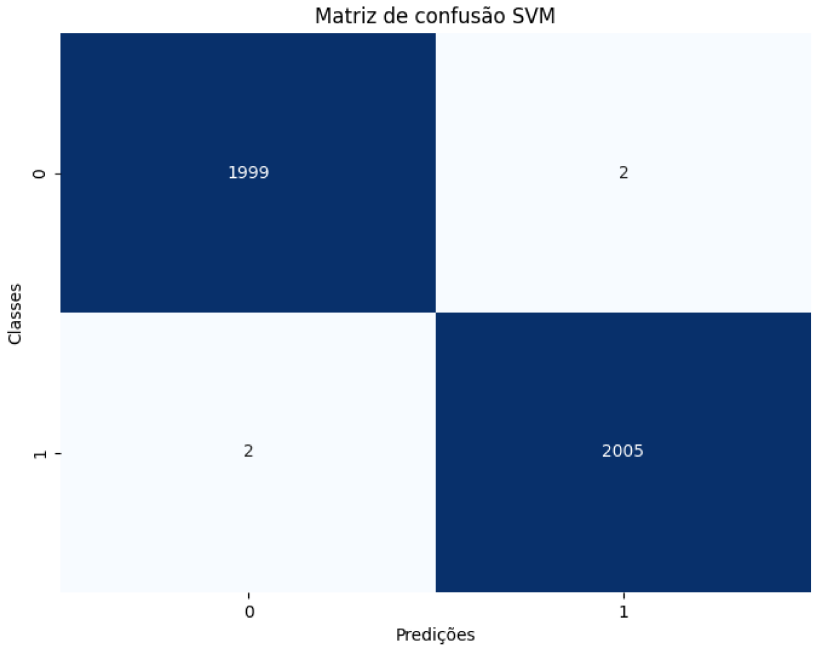

Figura 12 – Matriz de Confusão - SVM

Fonte: Autor

4.3 Seleçao dos Atributos

Com o objetivo de reduzir a dimensionalidade do conjunto de dados e aprimorar o desempenho dos modelos, foram realizadas a técnica de importância dos atributos(*Feature Importance*) e SHAP para analisar a contribuição individual de cada variável.

4.3.1 Resultados do processo de *Feature Importance* com Árvore de Decisão e SHAP

A partir da árvore gerada pelo modelo no *baseline*, exposta na Figura 13, pode-se inferir que o atributo *svcscans.nservices* possui mais influência nas decisões do modelo, atuando como ponto principal de separação entre amostras maliciosas ou benignas. Este atributo refere-se à quantidade de processos ativos no momento da análise da memória, sendo assim, é um atributo valioso que pode indicar atividade maliciosa — mas principalmente quando associada a outras métricas da memória pode ter uma melhora significativa em sua assertividade (AZMOODEH; MIANI; LASHKARI, 2020). Assim como pode ser observado na Figura 13, a árvore gerada analisa vários atributos além deste para definir se a atividade é de fato maliciosa ou não.

Baseado na análise da importância dos atributos na Árvore de Decisão, podemos notar **6** outros atributos de extrema importância para a decisão do modelo, a) svcscan.process_services, b) handles.nevent, c) handles.nthread, d) handles.avg_handles_per

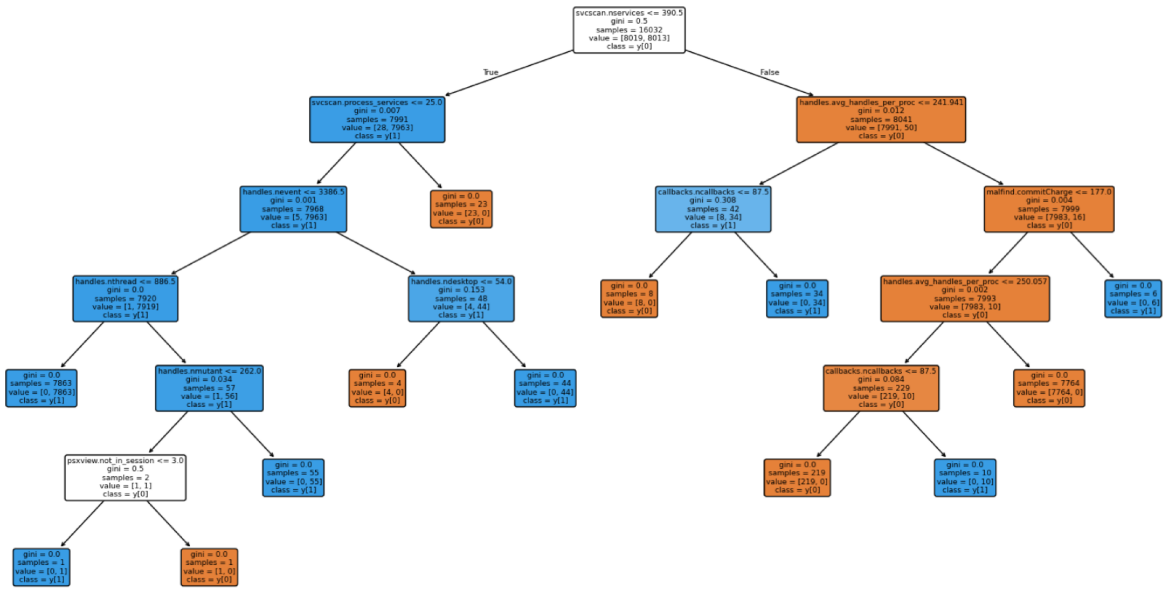

Figura 13 – Árvore Gerada pelo modelo treinado a partir do *Baseline*

Fonte: Autor

_proc, e) callbacks.ncallbacks, f) malfind.commitCharge.

Após a primeira etapa da otimização dos atributos, tornou-se essencial compreender de forma mais aprofundada como esses atributos influenciam as decisões dos algoritmos de classificação. Para aprimorar a transparência do processo decisório, foi aplicada a técnica SHAP, que permite atribuir valores de importância a cada atributo. A utilização do SHAP neste contexto não apenas proporciona uma compreensão mais clara das decisões do modelo, mas também fortalece a confiança na sua capacidade de distinguir entre comportamentos benignos e maliciosos.

Após analisar a Figura 14, observamos que os atributos *svcscan.process_services*, *handles.nevent*, *svcscan.nservices*, *handles.avg_handles_per_proc*, *callbacks.ncallbacks*, *dll-list.avg_dlls_per_proc*, *malfind.ninjections*, *ldrmodules.not_in_init_avg* e *handles.n-directory* influenciam positivamente as previsões do modelo. Esses atributos apresentam valores majoritariamente positivos no eixo horizontal, indicando sua significativa contribuição para as decisões do modelo. A distância dos pontos em relação ao centro reforça esse impacto.

4.4 Resultado do treinamento dos modelos otimizados

Com base na análise detalhada da Árvore de Decisão e dos valores SHAP feita na seção 4.3, todos os modelos foram reavaliados para verificar a perda, acurácia e tempo de inferência de cada um. E nesta seção serão expostos os resultados obtidos de cada um dos novos modelos gerados.

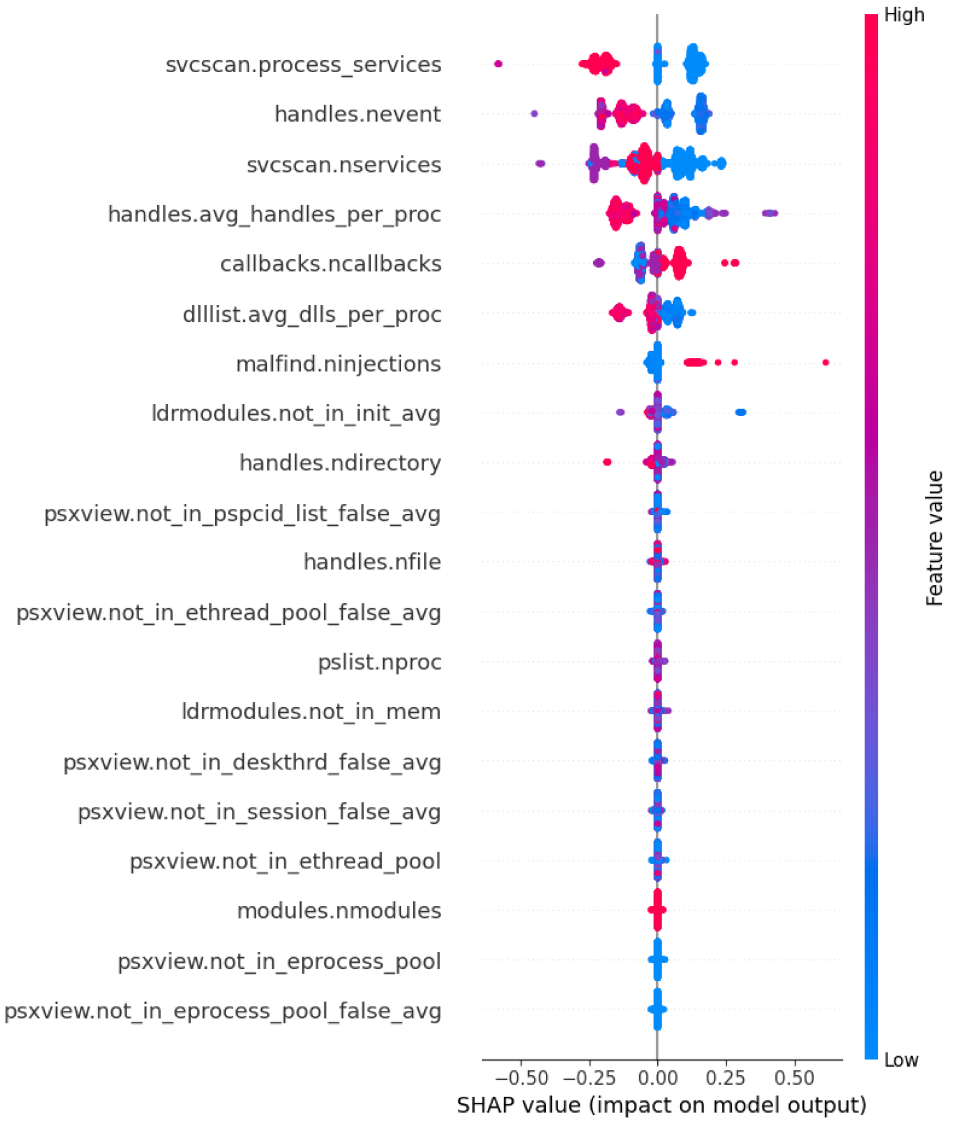

Figura 14 – Valores SHAP usando Árvore de Decisão

4.4.1 Rede Neural

Os resultados do novo modelo na Figura 15 demonstraram que o modelo manteve um desempenho equiparável com sua variante do *baseline* — demonstrando uma precisão muito alta ao ponderar que os números de perda estão variando na terceira casa decimal. Ou seja, não somente uma acurácia semelhante, como também o mesmo comportamento, denotando um aumento de acurácia e diminuição da perda ao longo das épocas.

A matriz de confusão apresentada na Figura 16 reforça a alta assertividade do modelo com **1973** verdadeiros negativos, **2004** verdadeiros positivos, **25** falsos negativos e apenas **6** falsos positivos. Nesse sentido, o comportamento da curva de acurácia em conjunto com os falsos negativos demonstram que o modelo obteve forte capacidade de predição com baixa taxa de falsos alarmes e erros de omissão. Com uma diferença pequena em relação a variante do *baseline*, pode-se considerar que esse modelo é bastante robusto

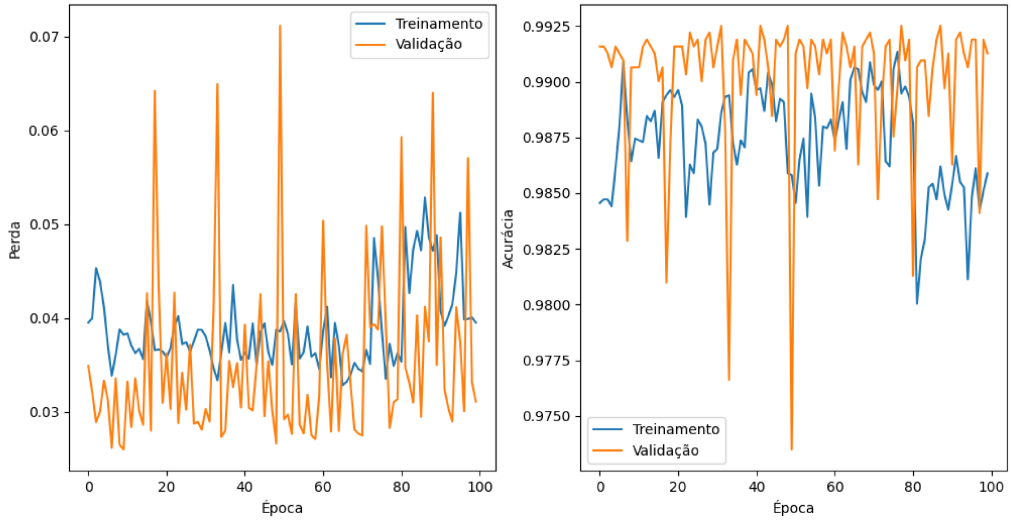

Figura 15 – Curva de Aprendizado - Acurácia - Rede Neural - Otimizado

Fonte: Autor

e não sofreu com a redução das *features*.

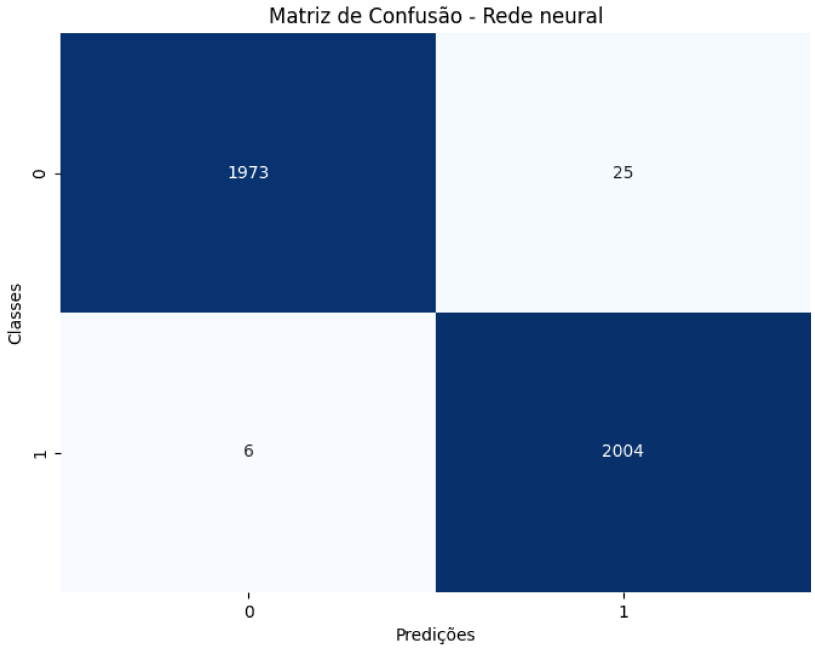

Figura 16 – Matriz de Confusão - Rede Neural - Otimizado

Fonte: Autor

4.4.2 Árvore de Decisão

A Árvore de Decisão otimizada apresentou o mesmo comportamento do modelo gerado no *baseline*. Como é possível notar na figuras Figuras 17 e 18, o modelo obteve uma acurácia de **100%** em treinamento e uma ótima conversão nas curvas de validação conforme o conjunto de treinamento aumenta — indicando melhora na capacidade de generalização com mais dados e reduzindo o impacto do *overfitting*.

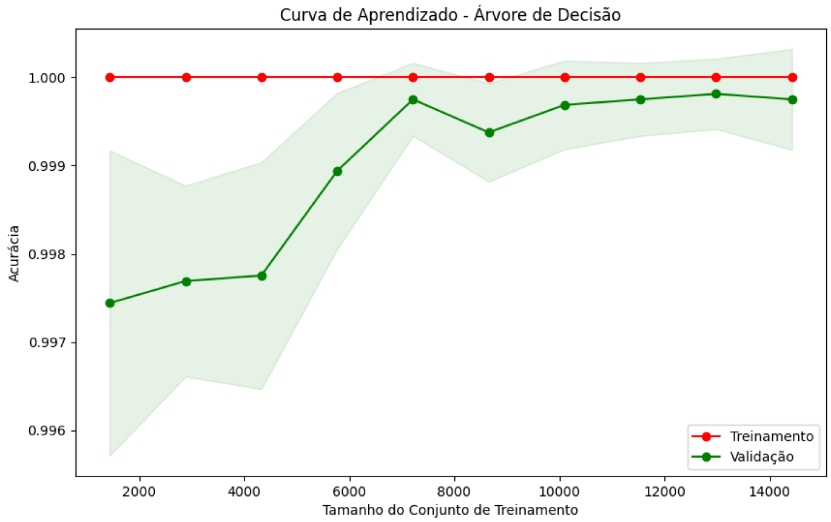

Figura 17 – Curva de Aprendizado - Acurácia - Árvore de Decisão - Otimizado

Fonte: Autor

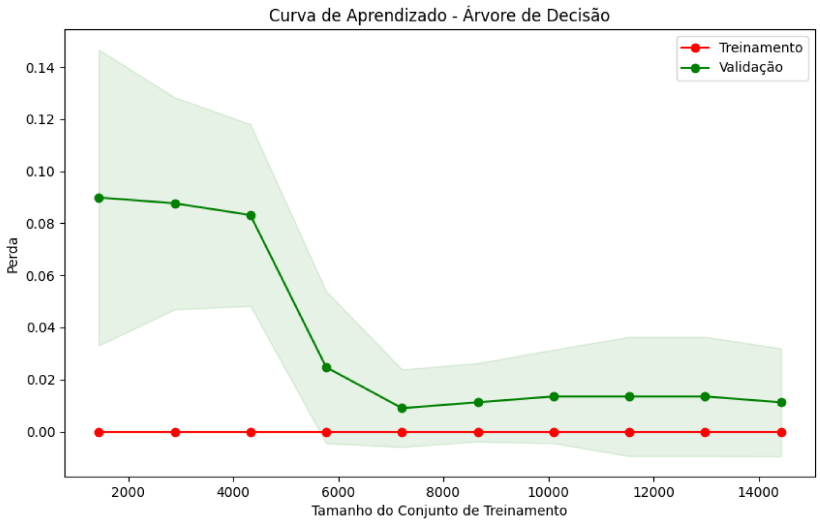

Figura 18 – Curva de Aprendizado - Perda - Árvore de Decisão - Otimizado

Fonte: Autor

Analisando a matriz de confusão na Figura 19, em conjunto com as curvas de perda e acurácia, podemos concluir que a matriz apresentou uma performance perfeita

para o modelo, com **100%** de precisão. Com falsos negativos e falsos positivos zerados, o modelo foi impecável em classificar os dados do conjunto de validação, com **100%** de precisão e sensibilidade. Sendo assim, o modelo não teve problemas com a remoção das *features*, indicando uma boa seleção de atributos.

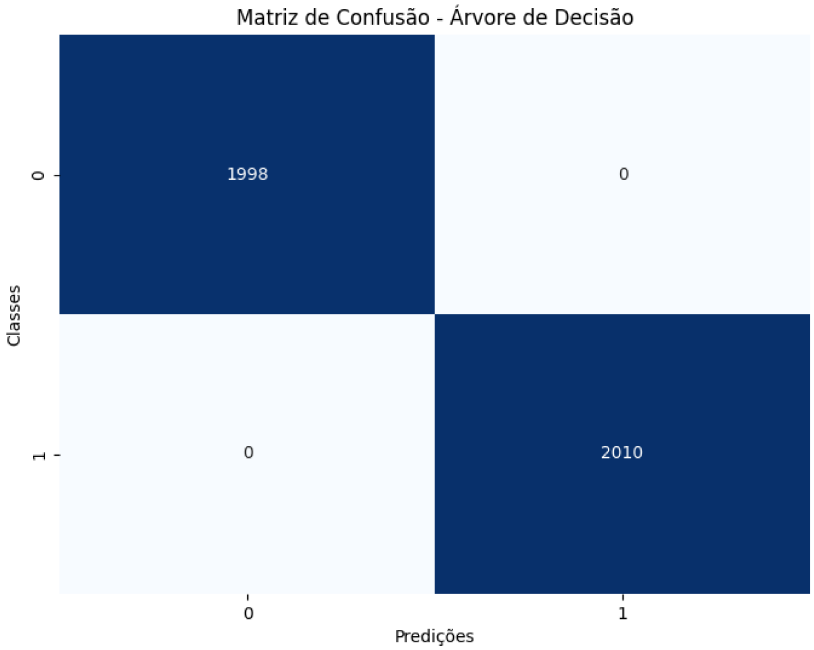

Figura 19 – Matriz de Confusão - Árvore de Decisão - Otimizado

Fonte: Autor

4.4.3 Naive Bayes

No gráfico de curva de aprendizado, com a acurácia do modelo na Figura 20, além uma acurácia elevada em treinamento e validação, foi observada também uma boa estabilidade; sugerindo um bom aprendizado do modelo no treinamento e uma generalização consistente. Ademais, as variações ou intervalos de confiança são muito estreitas, considerando a faixa de variação do eixo da acurácia, variando na terceira casa decimal. Em razão da diferença entre as curvas de treinamento e validação ser bem curta, o modelo não levanta suspeitas de *underfitting* ou *overfitting*, além de demonstrar uma performance balanceada.

O gráfico das curvas em relação à perda, na Figura 21, também demonstra uma boa consistência com níveis de perda muito baixos — indicadores que já iniciam baixos variam e convergem para um valor ainda menor. Com as linhas de validação e treinamento bem próximas, denota-se uma boa capacidade de aprendizado.

A matriz de confusão mostrada na Figura 22 confirma a excelente generalização do modelo, com baixos valores de **falsos positvos(6)** e **falsos negativos(20)**. Logo,

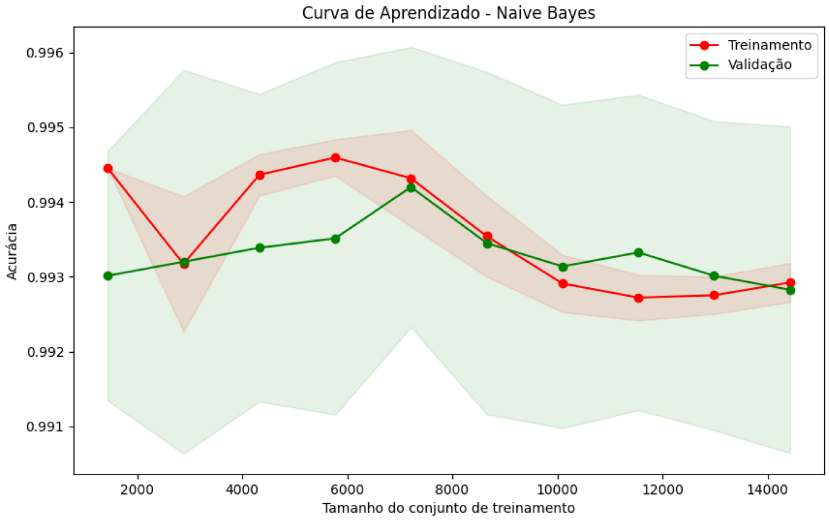

Figura 20 – Curva de Aprendizado - Acurácia - Naive Bayes - Otimizado

Fonte: Autor

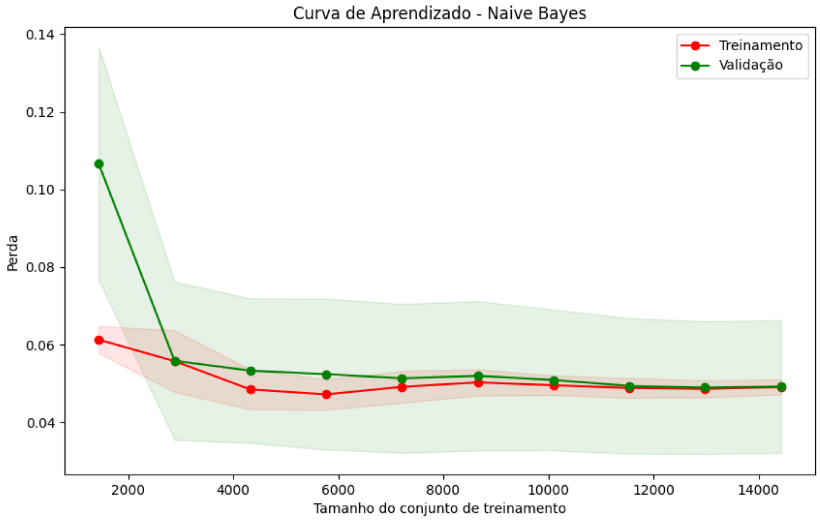

Figura 21 – Curva de Aprendizado - Perda - Naive Bayes - Otimizado

Fonte: Autor

demonstra ser um modelo bem eficiente e com desempenho sólido.

4.4.4 SVM

Na curva de acurácia do modelo representada pela Figura 23, a acurácia de treinamento começa alta e se mantém estável, indicando que o modelo está aprendendo consistentemente com os dados. Além disso, o fato de a acurácia não chegar a **100%** sugere que o SVM apresenta uma boa regularização e não super ajustado ao conjunto de treinamento. A acurácia de validação seguiu uma tendência semelhante à do treinamento,

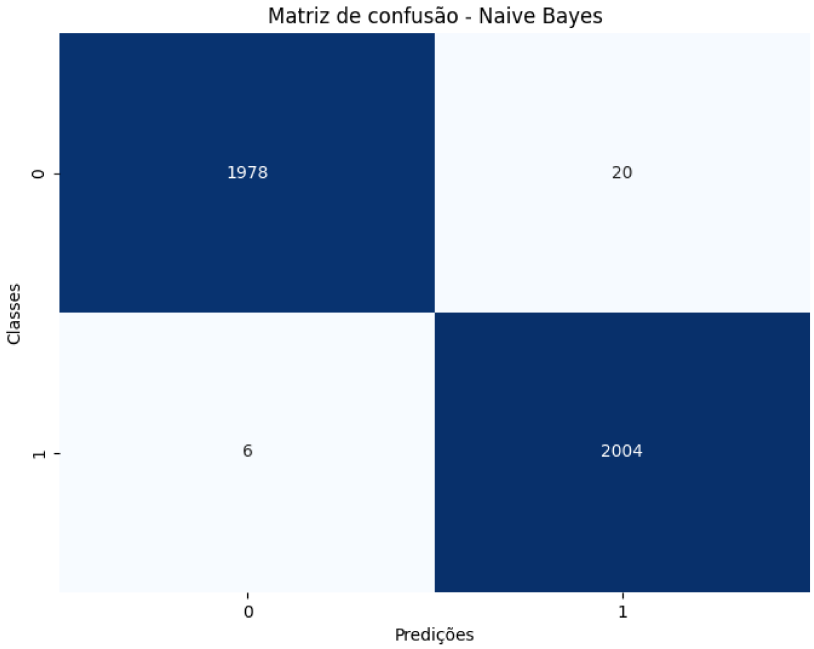

Figura 22 – Matriz de Confusão - Naive Bayes - Otimizado

Fonte: Autor

começando de um valor mais baixo e convergindo de forma estável para um valor mais alto, de acordo com o aumento do número de amostras. A proximidade entre as linhas evidencia que o modelo não apresentou *overfitting* ou *underfitting*, apontando que o modelo está generalizando bem.

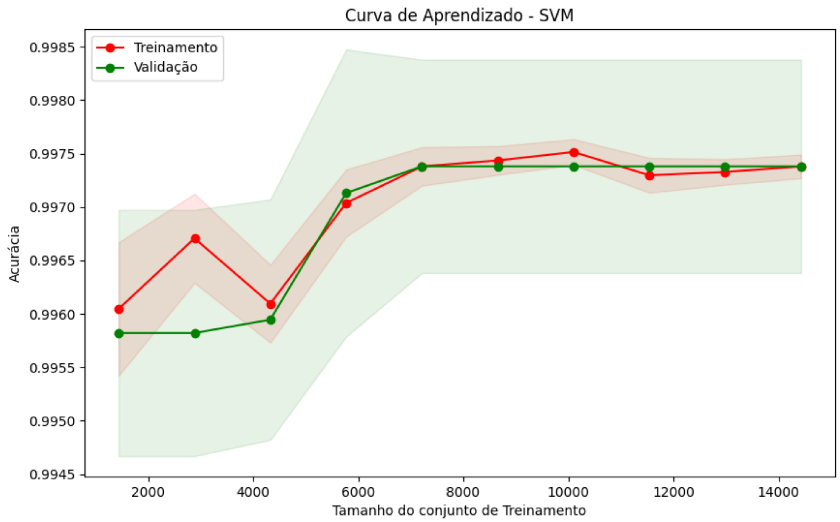

Figura 23 – Curva de Aprendizado - Acurácia - SVM - Otimizado

Fonte: Autor

A curva de perda do modelo na Figura 24 apresentou um bom desempenho, com

uma perda relativamente alta no início, mas que diminui de forma significativa à medida que o conjunto de dados aumenta — indicando que o modelo está aprendendo a se ajustar ao dado. Porém, é importante notar um leve aumento na perda quando o tamanho do conjunto de treinamento chega próximo a **8000** amostras, o que pode indicar padrões variados nas amostras. No entanto, não há sinais claros de *overfitting*, visto que a linha de validação se manteve bem estável, com uma tendência de queda conforme o número de amostras aumenta — ademais a diferença entre as linhas de treinamento e validação é pequena.

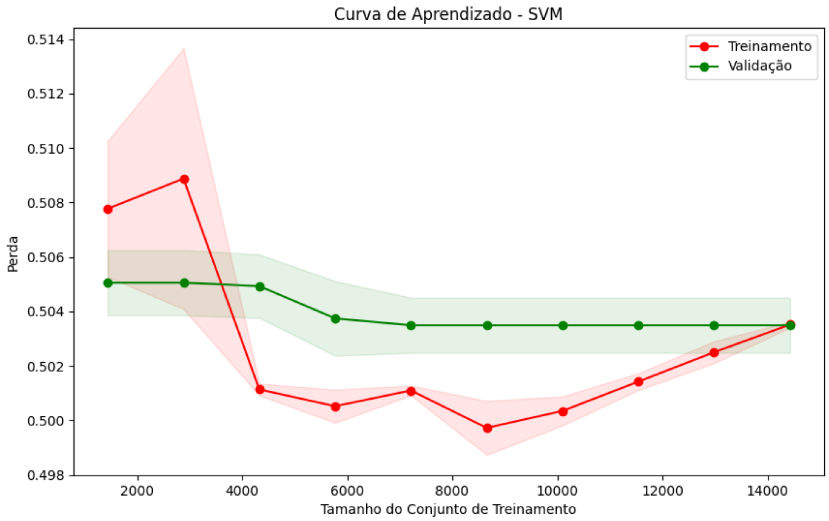

Figura 24 – Curva de Aprendizado - Perda - SVM - Otimizado

Fonte: Autor

A matriz de confusão do modelo na Figura 25 demonstrou uma performance altamente confiável, com um número muito baixo de **falsos positivos(1)** e **falsos negativos(13)**. A precisão do SVM foi ótima, podendo ser considerado muito robusto para identificar ambos os casos, tanto positivos como negativos.

4.5 Análise do Tempo de Inferência

Todos os modelos foram submetidos a uma análise do tempo de inferência, a fim de aprofundar-se no quesito eficiência e entender quais deles trabalham melhor o tempo de predição dos dados. Nessa seção, serão apresentados os comparativos de tempo de inferência unitário e também em bloco.

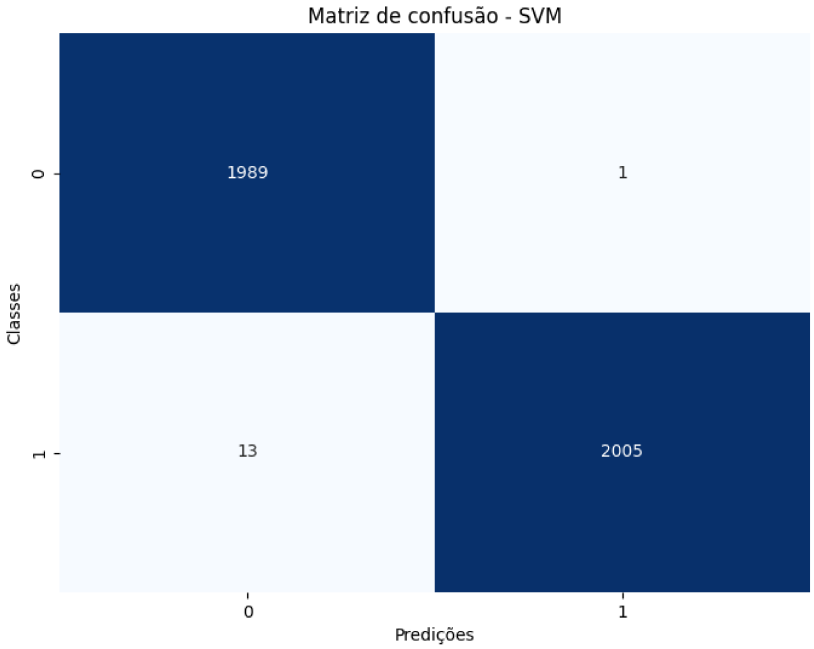

Figura 25 – Matriz de Confusão - SVM - Otimizado

Fonte: Autor

4.5.1 Comparativo de tempo por instância individual e tempo de execução em bloco

A Tabela 2 apresenta os tempos de inferência para os modelos em duas abordagens distintas: execução unitária (uma instância por vez) e execução em bloco (**100** instâncias processadas em conjunto).

A Tabela 2 apresenta os tempos de inferência para as duas abordagens propostas no capítulo **3** deste trabalho: execução unitária(uma amostra por vez) e execução em bloco(**100** amostras processadas em conjunto).

Modelos	**Benigno**		**Spyware**	
—	**Tempo Unitário**	**Tempo em Bloco**	**Tempo Unitário**	**Tempo em Bloco**
Rede Neural	161ms	159ms	151.3ms	158ms
Árvore de Decisão	25.5ms	22.4ms	23.3ms	25.5ms
Naive Bayes	27.9ms	21.5ms	23.9ms	23.6ms
SVM	25.5ms	22.4ms	19.6ms	24.6ms

Tabela 2 – Comparativo de inferência individual e em bloco

Fonte: Autor

Os resultados mostraram que o tempo de inferência para a execução em bloco é consistentemente menor ou semelhante ao tempo unitário para todos os modelos — ou seja, o mesmo tempo de execução de **1** para **100**. Este comportamento era exatamente o esperado, devido ao fato de que o processamento em bloco minimiza a sobrecarga associada

às chamadas repetidas ao método de predição.

Ao analisar a tabela, nota-se que o modelo **Rede Neural** apresentou os maiores tempos de inferência em ambas as abordagens, sendo, então, o mais adequado para cenários em que a eficiência computacional não é um requisito crítico. Por outro lado, os modelos **Árvore de Decisão**, **Naive Bayes** e **SVM** apresentaram tempos significativamente menores, com destaque para **SVM** e **Naive Bayes** — os quais apresentaram alta eficiência computacional, especialmente na execução em blocos.

Esses resultados indicam que, se tratando de aplicações em tempo real ou com grande volume de dados, modelos como **Naive Bayes** e **SVM** são mais viáveis devido a sua velocidade na inferência.

4.5.2 Comparativo de tempo para diferentes tamanhos de bloco

As Tabelas 3 e 4 destacam a performance de cada modelo para conjuntos de amostras de diferentes tamanhos, variando entre **100**, **200**, **300** amostras, analisando separadamente os tempos de inferência para dados benignos e *spywares*

Modelos	**Quantidade de Amostras**		
—	**100**	**200**	**300**
Rede Neural	162.2ms	189.8ms	180.7ms
Árvore de Decisão	25.5ms	24ms	23.5ms
Naive Bayes	21.4ms	21.6ms	24.5ms
SVM	21.5ms	28.2ms	29.2ms

Tabela 3 – Tempo de Inferência por Modelo por tamanho do conjunto de dados - Benigno

Fonte: Autor

Modelos	**Quantidade de Amostras**		
—	**100**	**200**	**300**
Rede Neural	160.7ms	171.8ms	179.7ms
Árvore de Decisão	19.8ms	20ms	21.3ms
Naive Bayes	21.5ms	21.5ms	21.6ms
SVM	20.5ms	27.5ms	31ms

Tabela 4 – Tempo de Inferência por Modelo por tamanho do conjunto de dados - Spyware

Fonte: Autor

Os tempos de inferência dos modelos para amostras benignas demonstraram um comportamento já esperado, mantendo seus tempos estáveis ou até mesmo diminuindo conforme o número de amostras aumenta — notadamente os modelos **Árvore de Decisão**, **Naive Bayes** e **SVM** —, refletindo a eficiência desses algoritmos em lidar com os diferentes volumes de dados. Se tratando do tempo de inferência para amostras de *spyware*, todos os modelos exceto a **Rede Neural** mantiveram uma constância nos valores ou até mesmo uma redução no tempo de inferência, como é o caso do **SVM** — o

modelo apresentou um aumento significativo no tempo de inferência com o aumento do tamanho dos blocos (de **162.2ms** para **180.7ms**), o que sugere uma maior complexidade do modelo para lidar com os diferentes padrões presentes nas amostras.

A análise de tais resultados sugere que a seleção do modelo precisa levar em consideração, primordialmente, o volume de dados. Nos cenários com alta frequência de amostras, os algoritmos como **SVM** e **Árvore de Decisão** oferecem um desempenho melhor e uma boa consistência em termos de tempo de resposta.

5 Análise de Resultados

A fim de entender com mais profundidade os resultados obtidos ao final deste trabalho, neste capítulo será analisado um aglomerado dos principais dados colhidos e análises feitas no capítulo 4. Os resultados dos modelos foram analisados com base em métricas quantitativas e qualitativas: inicialmente, foi analisada a performance bruta dos algoritmos sobre os dados originais e, posteriormente, após a aplicação de técnicas de seleção de atributos em conjunto, com **SHAP** para entender como cada atributo influencia na predição dos modelo. Também foi medido o tempo médio de inferência de cada modelo, utilizando execução individual e em bloco, para comparar a viabilidade de cada abordagem em contextos com restrição de recursos computacionais.

					Matriz de Confusão			
					Classe 0		Classe 1	
Processo	Modelos	Número de Atributos	Acurácia	K-Fold	Verdadeiros Negativos	Falsos Positivos	Falsos Negativos	Veradeiros Positivos
Baseline	Rede Neural	55	0,9935	0,9962	1991	16	10	1991
	Árvore de Decisão		1,0000	0,9996	2001	0	0	2007
	Naive Bayes		0,9900	0,9911	1972	11	29	1996
	SVM		0,9990	0,9989	1999	2	2	2005
Feature Importance e SHAP	Rede Neural	9	0,9923	0,9891	1973	6	25	2004
	Árvore de Decisão		1,0000	0,9996	1998	0	0	2010
	Naive Bayes		0,9933	0,9930	1978	6	20	2004
	SVM		0,9965	0,9974	1989	13	1	2005

Figura 26 – Tabela de Resultados

Fonte: Autor

A tabela de resultados na Figura 26 evidencia o impacto significativo do processo de otimização dos modelos, promovido pela técnica de *Feature Importance* e análise do SHAP, na performance dos modelos em detecção de *spyware*. Nota-se que, embora a otimização tenha causado ligeira diminuição da acurácia de modelos como **Rede Neural** e **SVM**, a seleção dos atributos resultou em mais equilíbrio para a classificação, refletido por um número menor de falsos positivos e maior eficiência computacional. Por outro lado, o modelo Naive Bayes apresentou ganhos substanciais tanto na acurácia quanto no balanceamento das classes, beneficiando-se do processo de seleção dos atributos. Por fim, a Árvore de Decisão manteve resultados considerados perfeitos antes e depois da otimização, indicando insensibilidade em relação à redução dos atributos. De maneira geral, a seleção dos atributos demonstrou ser uma estratégia eficaz, reduzindo a complexidade dos modelos e não comprometendo sua capacidade de classificação.

As análises de tempo de inferência apresentadas na seção 4.5 forneceram *insights*

fundamentais sobre a eficiência computacional dos modelos testados e sobre as características observadas em diferentes cenários de execução. Um dos achados mais relevantes foi a expressiva vantagem da execução em bloco sobre a execução unitária, demonstrando uma redução substancial no tempo total de inferência à medida que o número de amostras aumentava no processamento simultâneo.

Este fenômeno pode ser explicado pelas otimizações presentes nos *frameworks*. A documentação do **TensorFlow** cita que a inferência em bloco permite melhor alocação de recursos computacionais, aproveitando a paralelização e minimizando chamadas individuais ao *kernel*. Outro fator relevante para justificar o melhor desempenho na execução em bloco é a eliminação da sobrecarga causada pelo *warm-up*. Cada framework passa por um período inicial de configuração, onde recursos como *buffers* de memória, *threads* e processos são alocados. Durante a execução unitária, esse processo ocorre repetidamente, enquanto que em bloco, apenas uma vez para todo o conjunto de dados.

Em resumo, considerando o hardware e as configurações em que os testes realizados, podemos confirmar que a execução em bloco não apenas melhora a escalabilidade dos modelos de aprendizado de máquina, como também reduz significativamente a carga computacional imposta ao sistema, proporcionando inferência mais rápida e eficiente. Essa otimização é crucial para aplicações que lidam com grandes volumes de dados, garantindo respostas mais ágeis sem comprometer a precisão dos modelos.

Nesse contexto, é possível observar que os modelos **Árvore de Decisão**, **Naive Bayes** e **SVM** tendem a apresentar tempos de inferência estáveis, tanto para dados benignos quanto para *spyware*, pois são baseados em cálculos relativamente simples como as **verificações condicionais** da Árvore de Decisão. Essa estabilidade independe do tamanho do bloco, porém, conforme o tamanho do conjunto aumenta, há uma melhora devido às otimizações internas de processamento. Contudo, a **Rede Neural**, que envolve operações mais complexas como **cálculos matriciais** ou **propagação através de múltiplas camadas**, resultou em uma maior variação de tempo: inicialmente, demonstrou um tempo mais longo à medida que o número de amostras aumenta; todavia, conforme o número volta a crescer, há uma conversão para tempos mais baixos — o que pode estar ligado às otimizações internas dos *frameworks* e a complexidade do modelo que se adapta melhor com conjuntos de dados maiores.

Os resultados obtidos mostram que a escolha do modelo precisa levar em consideração o tipo de aplicação e as restrições computacionais envolvidas. Para cenários de alto desempenho, os modelos mais simples como **Árvore de Decisão**, **Naive Bayes** e **SVM** são ideais; enquanto para modelos mais complexos, o modelo de **Rede Neural** pode ser mais adequado em cenários onde a precisão e a capacidade de lidar com características em detalhes seja prioridade. Sendo assim, a análise de tempo de inferência reforça a necessidade de manter o equilíbrio entre complexidade e desempenho na escolha do modelo

para a detecção de *spyware*.

6 Conclusão

Este trabalho apresentou sucesso ao buscar a construção de um modelo preditivo otimizado capaz de detectar *spyware* através da análise da memória volátil do sistema, com quatro modelos preditivos desenvolvidos, treinados, testados e validados. Por meio dos resultados obtidos pela metodologia proposta, nota-se a eficiência dos modelos desenvolvidos, com acurácias altíssimas e tempo de resposta baixo. A seleção dos atributos — etapa fundamental para o projeto — demonstrou um grande potencial ao utilizar técnicas de aprendizado de máquina e inteligência artificial explicável para otimização dos modelos, aprimorando a precisão, a confiabilidade e a clareza dos métodos de detecção.

Os resultados obtidos demonstraram que a redução dos atributos não teve impacto significativo nos modelos. Em alguns casos, foi observada uma melhora no balanceamento entre falsos positivos e falsos, denotando a importância da escolha e otimização dos atributos para melhorar o desempenho dos modelos de ML — possibilitando reduzir drasticamente o número de *features* e obter uma performance semelhante ou ainda melhor. Quando analisados em conjunto com os resultados do tempo de inferência, é possível entender que os algoritmos seguem uma tendência de aumento de precisão e diminuição no tempo de inferência conforme o número de amostras aumenta, o que reforça a necessidade de equilibrar a precisão e a velocidade. Ou seja, escolhendo o modelo ideal considerando as demandas específicas de cada aplicação. Com isso, a pesquisa desenvolvida agrega significativamente para o entendimento do modelo ideal para cada cenário a ser utilizado, o que viabiliza a construção de soluções robustas, confiáveis e eficientes na detecção de *spyware*.

7 Trabalhos Futuros

Analisando os resultados e as limitações que foram observadas ao longo deste trabalho, diversas possibilidades de continuação podem ser exploradas em estudos futuros. Destarte, este capítulo apresenta propostas que visam expandir, aprofundar ou adaptar a abordagem adotada neste projeto.

7.1 Avaliação de outras técnicas de Interpretabilidade

O presente trabalho utilizou-se do método SHAP como técnica principal de explicabilidade. Como proposta, sugere-se a avaliação com outras técnicas de *Explainable Artificial Intelligence (XAI)* — como *Local Interpretable Model-agnostic Explanations (LIME)*, *Permutation Importance* ou *Deep Learning Important FeaTures (DeepLIFT)* — para fins de comparação, avaliação e criação de *insights* para qual técnica de XAI utilizar de acordo com o cenário — visto que tais técnicas podem oferecer diferentes perspectivas de interpretação e fornecer maior confiabilidade para os dados de estudo em segurança.

7.2 Aplicação em Tempo Real

Este trabalho parte da análise de um *dataset* proveniente de um experimento. Uma proposta relevante seria entender, na prática, como modelos desenvolvidos a partir de um *dataset* estático se comportam com os dados colhidos em tempo real da memória ou de *honeypots*. Algo que traria desafios diferentes, como a otimização do uso de recursos e a necessidade de respostas, aproximando o modelo de um sistema de detecção operável em tempo real em ambientes corporativos, industriais ou de uso cotidiano.

7.3 Implementação em Ambientes com Restrições de Recursos

Ampliando ainda mais a proposta da seção 7.2, essa proposta visa explorar a adaptação dos modelos em dispositivos embarcados ou sensores de borda, que possuem restrições de processamento e memória. Para isso, seria necessário considerar algoritmos mais leves, priorizando eficiência sem comprometer a acurácia.

8 Artefatos

Os artefatos produzidos ao longo desse trabalho estão presentes em um repositório github. Link do repositório

Referências

ABHIJNA, C.; AISHWARYA, A.; PRANAM, B. S.; RAGHURAMEGOWDA, S. et al. Malware detection using machine learning. In: IEEE. **2024 Second International Conference on Advances in Information Technology (ICAIT)**. India, 2024. v. 1, p. 1–5. Citado na página 12.

ABUALHAJ, M. M.; AL-SHAMAYLEH, A. S.; MUNTHER, A.; ALKHATIB, S. N.; HIARI, M. O.; ANBAR, M. Enhancing spyware detection by utilizing decision trees with hyperparameter optimization. **Bulletin of Electrical Engineering and Informatics**, v. 13, n. 5, p. 3653–3662, 2024. Citado 7 vezes nas páginas 12, 13, 18, 19, 20, 22 e 24.

ASLAN, A.; SAMET, R. A comprehensive review on malware detection approaches. **IEEE Access**, IEEE, v. 8, p. 6249–6271, 2020. Citado na página 16.

AZMOODEH, A.; MIANI, R.; LASHKARI, A. H. Volmemlyzer: A memory forensics analyzer for detecting advanced malware. In: **Proceedings of the 15th International Conference on Availability, Reliability and Security (ARES)**. Dublin, Ireland: ACM, 2020. p. 1–10. Citado na página 40.

BENSAOUD, A.; KALITA, J.; BENSAOUD, M. A survey of malware detection using deep learning. **Machine Learning with Applications**, Elsevier, v. 16, p. 100546, 2024. Citado 3 vezes nas páginas 13, 16 e 19.

CARRIER, B. **Detecting Obfuscated Malware Using Memory Feature Engineering**. New York, USA: Digital Forensics Press, 2021. Citado na página 17.

CHANDRASHEKAR, G.; SAHIN, F. Feature selection methods and applications. **Computers & Electrical Engineering**, v. 40, n. 1, p. 16–28, 2014. Citado na página 20.

CYBERSECURITY, C. I. for. **CIC-MalMem-2022 Dataset**. 2022. <https://www.unb.ca/cic/datasets/malmem-2022.html>. Accessed: 2024-04-14. Citado na página 25.

DENER, M. e. a. Malware detection using memory analysis data in big data environment. **Applied Sciences**, v. 12, n. 17, p. 8604, 2022. Citado na página 14.

EGELE, M.; KRUEGEL, C.; KIRDA, E.; YIN, H.; SONG, D. Dynamic spyware analysis. Carnegie Mellon University, 2007. Citado na página 17.

GUYON, I.; ELISSEEFF, A. An introduction to variable and feature selection. **Journal of machine learning research**, v. 3, p. 1157–1182, 2003. Citado na página 19.

JONASSON, D.; SIGHOLM, J. What is spyware. **TDDC03 Projects, Department of Computer and Information Science, Linkopings University, Sewden**, Citeseer, 2005. Citado 2 vezes nas páginas 17 e 18.

LEONEL, L.; MOLINOS, D.; MIANI, R. Comprehensive ransomware detection: Optimization of feature selection through machine learning algorithms and explainable

ai on memory analysis. In: **Anais do SBSeg**. São Paulo, Brasil: [s.n.], 2024. Citado 9 vezes nas páginas 14, 18, 19, 20, 21, 22, 23, 24 e 28.

MEZINA, A.; BURGET, R. Obfuscated malware detection using dilated convolutional network. In: IEEE. **2022 14th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)**. Brno, Czech Republic, 2022. p. 110–115. Citado 2 vezes nas páginas 21 e 22.

MOUSTAFA, N.; CREECH, G. Holistic feature selection for cyber threat detection in iot networks. **IEEE Transactions on Information Forensics and Security**, v. 14, p. 1325–1336, 2019. Citado na página 20.

NASSER, Y.; NASSAR, M. Toward hardware-assisted malware detection utilizing explainable machine learning: A survey. **IEEE Access**, IEEE, v. 11, p. 131273–131288, 2023. Citado na página 14.

QABALIN, M. K.; NASER, M.; ALKASASSBEH, M. Android spyware detection using machine learning: A novel dataset. **Sensors**, MDPI, v. 22, n. 15, p. 5765, 2022. Citado 2 vezes nas páginas 21 e 22.

RICHARD, L.; RIGAUD, S. **Pegasus: The Story of the World's Most Dangerous Spyware**. London, UK: Pan Macmillan, 2023. Citado na página 12.

VENTURES, C. **Cybercrime To Cost The World $8 Trillion Annually In 2023**. 2022. Accessed: 2025-05-06. Disponível em: <https://cybersecurityventures.com/cybercrime-to-cost-the-world-8-trillion-annually-in-2023/>. Citado na página 12.