Universidade Federal do Amazonas
Instituto de Computação
Programa de Pós-Graduação em Informática

Joner de Mello Assolin

MH-AutoML: Solução AutoML para o Domínio de *Malwares* Android

Manaus - Amazonas
2025

Joner de Mello Assolin

MH-AutoML: Solução AutoML para o Domínio de *Malwares* Android

Dissertação apresentada ao Programa de Pós-Graduação em Informática do Instituto de Computação da Universidade Federal do Amazonas como requisito parcial para obtenção do título de Mestre em Informática.

Orientador: Prof. Dr. Eduardo Luzeiro Feitosa
Coorientador: Prof. Dr. Diego Luis Kreutz

Manaus - Amazonas
2025

Ficha Catalográfica

Elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).

A849m Assolin, Joner de Mello
MH-AutoML: Solução AutoML para o Domínio de Malwares Android / Joner de Mello Assolin. - 2025.
73 f. : il., color. ; 31 cm.

Orientador(a): Eduardo Luzeiro Feitosa.
Coorientador(a): Diego Luis Kreutz.
Dissertação (mestrado) - Universidade Federal do Amazonas, Programa de Pós-Graduação em Informática, Manaus, 2025.

1. Android. 2. Malware. 3. AutoML. 4. Interpretabilidade. 5. XAI. I. Feitosa, Eduardo Luzeiro. II. Kreutz, Diego Luis. III. Universidade Federal do Amazonas. Programa de Pós-Graduação em Informática. IV. Título

---

Ministério da Educação
Universidade Federal do Amazonas
Coordenação do Programa de Pós-Graduação em Informática

FOLHA DE APROVAÇÃO

"MH-AUTOML: SOLUÇÃO AUTOML PARA O DOMÍNIO DE MALWARES ANDROID "

JONER DE MELLO ASSOLIN

Dissertação de Mestrado defendida e aprovada pela banca examinadora constituída pelos Professores:

Prof. Dr. Eduardo Luzeiro Feitosa - **Presidente**

Prof. Dr. Altair Olivo Santin - **Membro Interno**

Prof. Dr. Eduardo James Pereira Souto - **Membro Externo**

Manaus, 27 de agosto de 2025.

Documento assinado eletronicamente por **Eduardo James Pereira Souto**, **Professor do Magistério Superior**, em 27/09/2025, às 15:31, conforme horário oficial de Manaus, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015 .

Documento assinado eletronicamente por **Altair Olivo Santin**, **Usuário Externo**, em 28/09/2025, às 16:28, conforme horário oficial de Manaus, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Eduardo Luzeiro Feitosa**, **Professor do Magistério Superior**, em 01/10/2025, às 13:21, conforme horário oficial de Manaus, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015 .

Avenida General Rodrigo Octávio, 6200 - Bairro Coroado I Campus Universitário Senador Arthur Virgílio Filho, Setor Norte - Telefone: (92) 3305-1181 / Ramal 1193
CEP 69080-900, Manaus/AM, coordenadorppgi@icomp.ufam.edu.br

Dedico esta dissertação aos meus pais,
Rui Algusto Lopes Corrêa (in memoriam) e Janete Mello Corrêa,
que nunca mediram esforços para me dar uma boa educação
e me ensinaram, desde cedo, a importância dos estudos.

Entrega o teu caminho ao Senhor;
confia nEle,
e Ele tudo fará.

Salmos 37,5

Agradecimentos

Primeiramente, quero agradecer de coração ao meu orientador, o Prof. Dr. Eduardo Luzeiro Feitosa, e ao meu coorientador, o Prof. Dr. Diego Luis Kreutz. Eles foram incríveis, sempre estiveram lá para responder minhas milhares de perguntas e tiveram uma paciência infinita durante todo o desenvolvimento deste trabalho. Não sei o que teria feito sem eles.

Também não posso deixar de mencionar meus colegas do projeto *Malware Hunter*. A gente passou por muita coisa juntos, desde desafios técnicos até momentos de dúvida; e essa troca de experiência foi fundamental para chegarmos até aqui. Valeu, pessoal!

Por último, mas não menos importante, quero agradecer a todos que, de alguma forma, contribuíram e me apoiaram ao longo desse caminho. Às vezes, um simples incentivo faz toda a diferença, estou muito grato por ter pessoas incríveis na minha vida.

Resumo

Este estudo investiga o emprego de *frameworks de Automated Machine Learning* (AutoML) e propõe o desenvolvimento do MH-AutoML (*Malware Hunter Automated Machine Learning*), um *framework* específico projetado para detecção de malware Android. Essa solução representa uma evolução do premiado DroidAutoML, que recebeu reconhecimento como a segunda melhor ferramenta no Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg 2022). O cerne deste estudo reside em simplificar os processos cruciais inerentes ao pipeline de aprendizado de máquina, abrangendo limpeza de dados, engenharia de características, seleção de algoritmos, ajuste de hiperparâmetros e avaliação de modelos. O MH-AutoML está sendo meticulosamente desenvolvido com ênfase especial em personalização, transparência, interpretabilidade e depuração. Além disso, princípios de baixo acoplamento foram aplicados, melhorando substancialmente a organização da ferramenta, bem como sua capacidade de evolução e manutenção. O trabalho conclui com discussões que abrangem possíveis melhorias e direções futuras para o MH-AutoML. Essas considerações reconhecem a importância contínua de aprimoramento e inovação nesse domínio fundamental de pesquisa.

Palavras-chave: AutoML, Aprendizado de máquina, Framework, Detecção de *Malwares* Android.

Abstract

This study investigates the use of Automated Machine Learning frameworks (AutoML) and proposes the development of MH-AutoML (Malware Hunter Automated Machine Learning), a specialized framework designed for Android malware detection. This solution represents an evolution of the award-winning DroidAutoML, which was recognized as the second-best tool at the Brazilian Symposium on Information and Computational Systems Security (SBSeg 2022). The core of this study lies in simplifying the crucial processes inherent to the machine learning pipeline, including data cleaning, feature engineering, algorithm selection, hyperparameter tuning, and model evaluation. MH-AutoML is being meticulously developed with a special emphasis on customization, transparency, interpretability, and debugging. Additionally, principles of low coupling have been applied, substantially improving the tool's organization as well as its capacity for evolution and maintenance. The work concludes with discussions covering potential improvements and future directions for MH-AutoML. These considerations acknowledge the ongoing importance of refinement and innovation in this fundamental research domain.

Keywords: AutoML, Machine learning, framework, Android Malware Detection.

Lista de Figuras

Lista de Tabelas

Nomenclatura

ALR: Android Feature Selection Linear Regression-based

APKs: Arquivos de Pacotes Android

AutoML: Auto Machine Learning (Aprendizado de Máquina Automatizado)

BOHB Robust and Efficient Hyperparameter Optimization

EMBER: Endgame Malware BEnchmark for Research

GCP: Google Cloud Platform

ICA: Independent Component Analysis (Análise de Componentes Independentes)

KNN: K-Nearest Neighbors (Vizinhos mais Próximos K)

LDA: Linear Discriminant Analysis (Análise Discriminante Linear)

LightGBM: Light Gradient Boosted Machine

LIME: Explicações Locais Model-Agnostic

ML: *Machine Learning* (Aprendizado de Máquina)

MLOps: Machine Learning Operations (Operações de Aprendizado de Máquina)

MVC: Model-View-Controller, uma arquitetura de software

NNI Neural Network Intelligence

PCA: Principal Component Analysis (Análise de Componentes Principais)

POO: Programação Orientada a Objetos

RFE: Eliminação Recursiva de Características

RFG: Robust Feature Generation

SigAPI: Significant API Calls

SigPID: Significant Permission Identification

SRP: Princípio da Responsabilidade Única

SVD: Singular Value Decomposition (Decomposição em Valores Singulares)

SVM: Support Vector Machine (Máquina de Vetores de Suporte)

Sumário

Capítulo 1

Introdução

A plataforma Android mantém sua posição de liderança no mercado global de dispositivos móveis, com mais de 3 bilhões de aparelhos ativos em todo o mundo. A Google Play Store, principal canal de distribuição de aplicativos para Android e uma das maiores lojas de apps do planeta, oferecia cerca de 2,26 milhões de aplicativos no segundo trimestre de 2024 — número que, em seu auge, chegou a 4,67 milhões em 2021 (Statista, 2024). As estimativas apontam que o volume total de downloads de aplicativos na plataforma Android deverá ultrapassar 136 bilhões até o final de 2025 (Statista, 2025).

Essa imensa popularidade também impulsiona o desenvolvimento de aplicativos maliciosos (*malwares*) direcionados à plataforma. Além do amplo alcance da plataforma, a facilidade de distribuição desses aplicativos contribui para o aumento da ameaça: usuários de Android podem instalar softwares provenientes de lojas alternativas, fora do ecossistema oficial da Google Play, o que dificulta o controle e a detecção de conteúdos maliciosos. Apesar da magnitude e da importância da loja oficial, a segurança permanece um desafio significativo. Em 2023, mais de 600 milhões de downloads de aplicativos maliciosos foram registrados na Google Play Store (Kaspersky, 2023).

Neste cenário, a abordagem de detecção de *malwares* baseada em aprendizado de máquina (*Machine Learning* ou ML) se destaca ao capacitar a identificação de padrões que podem ser aplicados a novas variantes de *malwares*. No entanto, essa abordagem apresenta uma desvantagem substancial: a necessidade de conhecimento técnico abrangente, englobando conceitos de matemática, estatística e computação, aliados a um profundo entendimento do campo de atuação.

Para evitar ou diminuir essa exigência, a AutoML (*Automated Machine Learning*) emergiu como uma solução para simplificar e automatizar diversas etapas do processo. O objetivo é acelerar a disponibilização de modelos otimizados e reduzir a dependência

de conhecimento técnico especializado (Karmaker S. et. al., 2021).

Resumidamente, *frameworks* de AutoML permitem obter artefatos testáveis com uma complexidade e um custo de tempo menor em comparação com abordagens manuais. Em particular, eles reduzem substancialmente a curva de aprendizagem, uma vez que eliminam a necessidade de diversos conhecimentos específicos da área de aprendizado de máquina.

Entretanto, *frameworks* como AutoSklearn (Feurer et al., 2015a), GoogleAutoML, DataRobot, Qeexo, AutoGluon (Erickson et al., 2020), AutoWEKA (Thornton et al., 2013), H2OAutoML (LeDell e Poirier, 2020a), TPOT (Olson e Moore, 2016) e Auto-PyTorch (Zimmer et al., 2021) são tipicamente de aplicação geral, não sendo desenvolvidos com o objetivo de gerar modelos otimizados para domínios específicos, como detecção de *malwares*.

Frameworks de AutoML orientados a um domínio possuem o benefício de permitir o processamento especializado de dados, de modo a obter o máximo de informação relevante em relação ao domínio. Por exemplo, existem soluções de AutoML específicas para classificação de diagnósticos médicos (Alaa e Schaar, 2018; Tsamardinos et al., 2020), que obtém resultados melhores do que *frameworks* gerais de AutoML pelo fato de especializarem etapas importantes do *pipeline*. Até nosso conhecimento, não existe um framework de AutoML para o domínio de detecção de *malware* Android.

1.1 Problema e Motivação

A concepção inicial de um *framework* de AutoML para a detecção de *malware* Android surgiu no contexto do projeto de pesquisa Malware Hunter, com a construção de uma versão embrionária entre 2021 e 2022 Inicialmente, chamada de DroidAutoML, o *framework* obteve reconhecimento ao ser premiada como a segunda melhor solução no Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg 2022) (Assolin et al., 2022).

Contudo, durante a análise dos *frameworks* existentes, identificamos diversas lacunas tanto no conceito quanto na implementação prática das soluções de AutoML, especialmente no contexto de detecção de *malware* Android. Entre os principais pontos de melhoria, destacam-se a falta de transparência nos processos internos, a baixa interpretabilidade dos modelos gerados, a ausência de mecanismos eficazes para o versionamento de modelos e a limitada capacidade de controle sobre todo o ciclo de vida das soluções.

Observamos que tanto a DroidAutoML quanto outros *frameworks* similares falham

ao lidar com conjuntos de dados específicos do domínio Android, como o conjunto Androcrawl (SISTO, 2012). Essa falha deixa evidente que a etapa de pré-processamento dos dados é de suma importância para trabalhar em contextos específicos. Um exemplo prático ocorre quando esses *frameworks* tentam processar dados do tipo Object como float, o que resulta em falhas ao interpretar corretamente informações como assinaturas criptográficas (e.g., SHA256 ou MD5), que são essenciais para a identificação dos aplicativos.

Na versão atual do *framework*, essas assinaturas são automaticamente reconhecidas e utilizadas de forma apropriada, tanto para a remoção de registros duplicados quanto para a exclusão de atributos irrelevantes do tipo *Object*, como nomes de aplicativos ou *marketplaces*. Outro aspecto crítico é a ausência de suporte à interpretabilidade nos modelos gerados pelos *frameworks* avaliados, consequência direta da falta de um pré-processamento robusto e consciente do domínio.

Diante desse cenário, este estudo tem como propósito o desenvolvimento de um *framework* de AutoML voltado à detecção de *malwares* Android, fundamentado no DroidAutoML, mas com avanços significativos. O *framework* buscará abranger todas as etapas de um pipeline típico de aprendizado de máquina, com ênfase em interpretabilidade dos modelos, transparência nos processos, controle de versão e facilidade de uso.

1.2 Objetivo

O objetivo desta dissertação é desenvolver o MH AutoML (*Malware Hunter AutoML*), um *framework* de AutoML especializado na detecção de *malwares* Android. O projeto busca construir um pipeline automatizado que integra bibliotecas públicas e de código aberto, como *scikit-learn*[1], *MLflow*[2] e *Optuna*[3].

A biblioteca *scikit-learn* é amplamente utilizada para aprendizado de máquina, oferecendo implementações consolidadas de algoritmos de classificação, regressão e redução de dimensionalidade. Já o *MLflow* aplica conceitos de *MLOps* (*Machine Learning Operations*), que consistem em práticas para padronizar, automatizar e monitorar o ciclo de vida de modelos de aprendizado de máquina em ambientes de produção. Por fim, o *Optuna* é uma ferramenta especializada em otimização de hiperparâmetros, permitindo a busca eficiente por combinações que maximizem o desempenho dos modelos.

[1] https://scikit-learn.org
[2] https://mlflow.org
[3] https://optuna.org

Esse pipeline automatiza etapas essenciais do processo de aprendizado de máquina, incluindo extração e seleção de atributos, treinamento de modelos, otimização de hiperparâmetros e análise de desempenho. O propósito é oferecer uma solução eficiente, acessível e especificamente adaptada ao contexto da segurança digital.

Como objetivos específicos, pretende-se:

- Incorporar técnicas de interpretabilidade aos modelos gerados, utilizando métodos como SHAP e LIME, com o objetivo de fornecer explicações claras sobre os fatores que influenciam as decisões de classificação, visando maior transparência e apoio a análises forenses;
- Integrar práticas de MLOps ao ciclo de vida do *frameworks*, assegurando o rastreamento de experimentos, o versionamento de modelos e dados, e a reprodutibilidade dos resultados por meio de ferramentas como MLflow;
- Realizar experimentos de *benchmarking*, comparando o desempenho, a interpretabilidade e a transparência do MH-AutoML com *frameworks* AutoML de propósito geral, utilizando métricas apropriadas para o domínio de detecção de *malwares* Android;
- Propor e validar um modelo de avaliação de transparência e interpretabilidade em sistemas AutoML, com foco em domínios sensíveis como a segurança digital, estabelecendo critérios objetivos para comparação entre diferentes soluções.
- Criar um conjunto de dados abrangente e atualizado para o domínio de detecção de *malwares* Android, considerando a carência de bases representativas e recentes. Soares et al. (2021);

1.3 Contribuições Alcançadas

As principais contribuições deste trabalho são duas. A primeira é um *framework* especializado que entrega desempenho competitivo com soluções existentes, sem sacrificar interpretabilidade. Seu diferencial chave é integrar *explicabilidade nativa* com **MLOps robusto**, oferecendo não apenas alta acurácia, mas um ciclo de vida completo onde:cada predição é auditável, cada experimento é reproduzível e cada modelo é versionado com seus metadados de interpretabilidade. A segunda é um modelo de avaliação inédito para mensurar transparência e interpretabilidade em sistemas AutoML, estabelecendo novos padrões para a área.

Tabela 1.1: Contribuições em Pesquisa de Detecção de Malwares Android (2022-2024).

Ano	Contribuição	Tipo	Papel	Link
2022	**DroidAutoML**: Framework de AutoML para detecção de malwares Android *Prêmio*: 2ª melhor trabalho (SBSeg)	Framework	Autor principal	GitHub
	AdBuilder: Ferramenta de construção de datasets	Framework	Coautor	GitHub
	Avaliação de frameworks de AutoML em datasets de malware	Estudo Comparativo	Coautor	GitHub
	Análise de métodos de seleção de características	Estudo Metodológico	Coautor	GitHub
2023	**AMGenerator e AMExplorer**: Geração de metadados Android	Framework	Coautor	GitHub
2024	**MH-AutoML**: Framework com interpretabilidade e MLOps *Prêmio*: Artefato destaque (SBSeg)	Framework	Autor principal	GitHub
	MH-API: Solução escalável para detecção	Framework	Autor principal	–
	MH-1M: Dataset abrangente de malware Android	Dataset	Coautor	GitHub
	SigAPI AutoCraft: Seleção de características *Prêmio*: Artefato destaque (SBSeg)	Framework	Coautor	GitHub
	Análise exaustiva de métodos de seleção de características	Estudo Metodológico	Coautor	GitHub

A pesquisa também gerou um conjunto integrado de artigos que, em sinergia, avançaram o estado da arte em detecção automatizada de *malwares* Android. A Tabela 1.1 sintetiza esse ecossistema, destacando como cada componente – seja teórico (estudos metodológicos), prático (*frameworks*) ou de infraestrutura (*datasets*) – contribui para um ciclo completo de pesquisa e aplicação, desde a coleta de dados até a detecção em produção.

Os **estudos metodológicos** fornecem a base analítica para avaliação de técnicas de seleção de características em detecção de *malware* Android, realizando avaliações comparativas abrangentes que englobam desde métodos estatísticos consolidados (LASSO e PCA) até abordagens especializadas recentes (SigPID e JOWNDroid). Essas análises permitiram identificar o equilíbrio ótimo entre acurácia e desempenho computacional específico para o domínio de *malwares* Android (Rocha et al., 2024; Soares et al., 2022; Tschiedel et al., 2024).

No âmbito dos ***frameworks* desenvolvidos**, a evolução das soluções reflete diretamente os achados desses estudos. O *AdBuilder* (Vilanova et al., 2022) estabeleceu as bases para construção automatizada de datasets, enquanto seu sucessor *AMGenerator/AMExplorer* (Rocha et al., 2023) resolveu um desafio crítico identificado nas pesquisas: a geração dinâmica de conjuntos de dados atualizados e representativos. Paralelamente, o *DroidAutoML* (Assolin et al., 2022) (premiado em 2022) pioneiramente automatizou pipelines completos de ML, sendo posteriormente superado pelo *MH-AutoML* (Assolin et al., 2024), que incorporou lições cruciais dos estudos comparativos (Siqueira et al., 2022) - especialmente no que tange à transparência operacional e interpretabilidade dos modelos, dois gargalos anteriormente identificados.

O dataset MH-1M Bragança et al. (2024) constitui uma contribuição relevante, ao disponibilizar uma coleção abrangente e recente de amostras maliciosas para Android, contribuindo para mitigar a escassez de dados de qualidade identificada nas pesquisas iniciais Soares et al. (2021). Finalmente, o *MH-API* (de Mello Assolin et al., 2024) completa o ecossistema ao viabilizar a operacionalização escalável dessas soluções em ambientes de produção, fechando o ciclo entre pesquisa teórica e aplicação prática.

1.4 Organização da Dissertação

Esta dissertação está organizada da seguinte forma: o Capítulo 2 apresenta os conceitos fundamentais de *Machine Learning* (ML) e *Automated Machine Learning* (AutoML), com o objetivo de fornecer uma visão geral do tema; o Capítulo 3 discute os trabalhos relacionados, organizados em três categorias: estudos comparativos de *frameworks*, propostas de *frameworks* e pesquisas voltadas à interpretabilidade; o Capítulo 4 descreve o desenvolvimento da proposta, destacando a evolução do *framework*; o Capítulo 5 detalha a metodologia adotada, incluindo ambiente experimental, métricas e parâmetros utilizados; o Capítulo 6 apresenta e analisa os resultados obtidos; e, por fim, o Capítulo 7 reúne as conclusões e aponta possíveis direções para trabalhos futuros.

Capítulo 2

Conceitos Básicos

Este capítulo introduz os principais conceitos de aprendizado de máquina, AutoML e MLOps, com foco em suas aplicações no domínio de detecção de malwares Android. Esses fundamentos sustentam as escolhas metodológicas apresentadas nos capítulos seguintes.

2.1 Pipeline de Aprendizado de Máquina

O desenvolvimento de modelos de aprendizado de máquina segue um fluxo estruturado de etapas interdependentes, conhecido como pipeline de ML. Conforme definido por Mitchell (1997), este processo busca melhorar continuamente o desempenho do modelo (P) em tarefas específicas (T) através da experiência obtida dos dados (E). A Figura 2.1 apresenta visualmente este fluxo de trabalho, que vai desde o pré-processamento dos dados até a avaliação final do modelo.

2.1.1 Pré-Processamento de Dados

Essa etapa inicial tem como objetivo garantir a qualidade dos dados de entrada antes do treinamento dos modelos. Inclui atividades como identificação e tratamento de valores ausentes, duplicados e discrepantes (Kriegel et al., 2010). Técnicas como imputação estatística e remoção de *outliers* são empregadas para minimizar ruídos que possam comprometer o desempenho dos modelos. Além disso, os dados passam por normalizações e transformações, como padronização de escalas ou transformações não lineares (ex.: logarítmica), para adequar suas distribuições às necessidades dos algoritmos de aprendizado (Kriegel et al., 2010).

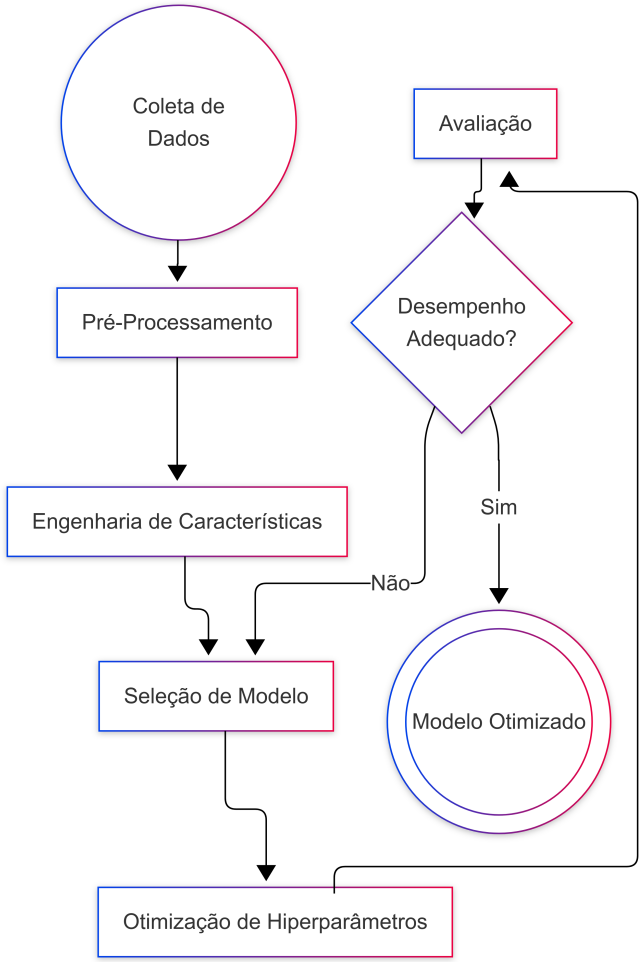


Figura 2.1: Fluxo convencional de desenvolvimento de modelos de aprendizado de máquina. Adaptado de (Truong et al., 2019).

2.1.2 Engenharia de Características

Nesta fase, o foco é transformar e selecionar os atributos mais relevantes (Domingos, 2012). Entre as técnicas de transformação, destacam-se a redução de dimensionalidade por meio de PCA (Jolliffe e Cadima, 2016), t-SNE (Maaten e Hinton, 2008) e UMAP (McInnes et al., 2018), e a aplicação de *autoencoders* (Hinton e Salakhutdinov, 2006). Para o tratamento de conjuntos desbalanceados, particularmente relevante na área de cibersegurança, destacam-se o método SMOTE (Chawla et al., 2002), variações como Borderline-SMOTE (Han et al., 2005) e ADASYN (He et al., 2008), além de métodos baseados em *ensemble* (Liu et al., 2009) e aprendizado por custo sensível

(Elkan, 2001). Por fim, a seleção de características abrange desde métodos simples baseados em filtros (Yu e Liu, 2003) até técnicas mais sofisticadas como RFE (Guyon et al., 2002) e regularização L1 (Tibshirani, 1996).

2.1.3 Seleção e Otimização de Modelos

Esta etapa é responsável por escolher o algoritmo de aprendizado mais adequado ao problema e por configurar seus hiperparâmetros (Probst et al., 2019). A seleção de modelos baseia-se em comparações entre diferentes abordagens, considerando fatores como desempenho, capacidade de generalização e custo computacional (Fernández-Delgado et al., 2014). São analisados algoritmos tradicionais, como *Random Forest* (Breiman, 2001) e *Gradient Boosting* (Friedman, 2001), e também técnicas mais recentes, como redes neurais profundas (LeCun et al., 2015) e SVMs (Cortes e Vapnik, 1995). Na otimização de hiperparâmetros, utilizam-se métodos modernos como busca bayesiana (Snoek et al., 2012), otimização por enxame de partículas (Kennedy e Eberhart, 1995) e meta-aprendizado (Feurer et al., 2015c), com o suporte de ferramentas como Hyperopt[1] e Optuna[2].

2.1.4 Avaliação e Interpretação

Na fase final do pipeline, os modelos são avaliados com base em métricas adequadas ao problema de detecção de *malware*, como AUC-ROC, precisão e *recall*. Uma validação robusta, utilizando técnicas como validação cruzada aninhada, garante que os modelos sejam generalizáveis para novos dados. Além disso, busca-se interpretar o comportamento dos modelos, identificando os fatores que mais influenciam suas decisões. Para isso, são empregadas técnicas de interpretabilidade como SHAP (Lundberg e Lee, 2017) e LIME (Ribeiro et al., 2016), que permitem analisar possíveis vieses e garantir a confiabilidade das previsões.

2.2 AutoML

A automação de tarefas específicas no contexto do aprendizado de máquina data dos anos 1990, mas o termo AutoML e a ideia de uma abordagem abrangente de automação ganharam destaque, principalmente, após 2010 (Truong et al., 2019), devido ao interesse e a inovação no campo do aprendizado de máquina.

[1] https://hyperopt.github.io/hyperopt/
[2] https://optuna.org/

Resumidamente, AutoML é um paradigma para automatizar o *pipeline* de aprendizado de máquina e reduzir o esforço manual envolvido com esse tipo de tecnologia, de modo a acelerar o desenvolvimento de modelos significativos e eficazes para resolver problemas de diferentes domínios (Karmaker S. et. al., 2021; Waring et al., 2020). A ideia é substituir, gradualmente, os requisitos e funções de humanos especialistas na técnica ou no problema no *pipeline* de aprendizado de máquina.

A automação de tarefas no campo do aprendizado de máquina apresenta os seguintes benefícios (Escovedo e Koshiyama, 2022; He et al., 2023; Zöller e Huber, 2021):

- **Economia de tempo:** a automação de tarefas de aprendizado de máquina permite que os usuários criem modelos mais rapidamente, sem a necessidade de passar horas ajustando parâmetros e selecionando modelos.
- **Aumento da eficiência:** a automatização de tarefas repetitivas e demoradas permite que os usuários concentrem seus esforços em tarefas mais importantes, como a análise e interpretação dos resultados.
- **Redução de erros:** a automação de processos pode reduzir a probabilidade de erros humanos, melhorando a precisão dos modelos gerados.
- **Maior acessibilidade:** um *frameworks* de AutoML pode permitir que usuários com pouco conhecimento em aprendizado de máquina possam criar modelos de alta qualidade
- **Melhoria na interpretabilidade:** a geração de modelos mais interpretáveis, pois a transparência e a explicabilidade são essenciais.
- **Aumento da inovação:** possibilidade de gerar novas soluções e obter conhecimentos relevantes.

2.2.1 Frameworks de AutoML

A Tabela 2.1 apresenta uma análise dos principais *frameworks* de AutoML encontrados na literatura e na prática atual.

A análise revela algumas lacunas no ecossistema atual, como poucos *frameworks* especializados. Foram encontrados apenas dois: AutoML4Clust para clustering e Darwin/NVIDIA TAO para visão computacional. Se por um lado, nota-se uma crescente participação de grandes empresas (AWS, Google, Microsoft, NVIDIA, Uber), por outro nota-se uma carência de soluções maduras para outros domínios específicos como NLP ou séries temporais.

Tabela 2.1: Lista de frameworks AutoML.

Framework	**Licença**	**Linguagem**	**CLI?**	**Aplicação**	**Empresa/Org.**
Auto-Keras	MIT	Python	Sim	Propósito geral	DATA Lab
Auto-Sklearn	BSD-3	Python	Sim	Propósito geral	Uni Freiburg
AutoGluon	Apache 2.0	Python	Sim	Propósito geral	AWS
H2O AutoML	Apache 2.0	Java	Sim	Propósito geral	H2O.ai
LightAutoML	Apache 2.0	Python	Sim	Propósito geral	Sberbank
TPOT	LGPL	Python	Sim	Propósito geral	UPenn
Ludwig	Apache 2.0	Python	Sim	Propósito geral	Uber
AutoML4Clust	MIT	Python	Não	Análise de clusters	Academia
FLAML	MIT	.NET	Não	Propósito geral	Microsoft
TransmogrifAI	BSD-3	Scala	Não	Propósito geral (dados estruturados)	Salesforce
Darwin	Proprietária	Python	Não	Visão computacional	V7 Labs
DataRobot	Proprietária	R	Não	Propósito geral	DataRobot Inc.
MLJAR	Proprietária*	Python	Sim	Propósito geral	MLJAR
Vertex AI	Proprietária	Python	Sim	Propósito geral	Google
NVIDIA TAO	Proprietária	Python	Sim	Visão computacional	NVIDIA

No âmbito de implementação, existe um predomínio de Python como linguagem principal (73% dos *frameworks*) e a disponibilidade de interfaces de linha de comando (CLI).

2.3 MLOps

Uma camada transversal ao *pipeline*, a aplicação do conceito de MLOps (*Machine Learning Operations*) estabelece os princípios de governança, monitoramento e automação contínua sobre o fluxo tradicional de desenvolvimento de modelos de ML. Enquanto um *pipeline* de AutoML foca na automação de etapas pontuais como seleção de algoritmos, ajuste de hiperparâmetros e engenharia de características, o MLOps atua como um meta-processo que engloba todo o ciclo de vida do modelo - desde o versionamento de dados até a implantação em produção e monitoramento pós-implantação.

O paralelo entre as duas abordagens pode ser explicitado em três níveis: (1) Na fase de experimentação, o AutoML automatiza a geração de modelos candidatos, enquanto o MLOps gerencia a reprodutibilidade desses experimentos através de versionamento de dados, código e artefatos; (2) Na transição para produção, o AutoML entrega modelos otimizados, mas o MLOps provê a infraestrutura para empacotamento (containerização), orquestração (Kubernetes) e *deploy* gradual (A/B *testing*); (3) Em operação, enquanto o AutoML não possui mecanismos nativos para monitoração, O AutoML não possui mecanismos nativos para monitoração. Já o MLOps implementa sistemas de detecção de *concept drift*, que ocorre quando a relação entre as variáveis de entrada e saída de

um modelo muda ao longo do tempo, reduzindo sua acurácia preditiva.

Essa simbiose mostra que AutoML e MLOps são complementares: o primeiro acelera a construção de modelos, o segundo garante que esses modelos gerem valor sustentável em ambientes produtivos. A arquitetura de MLOps normalmente incorpora estágios de AutoML como um componente especializado dentro de seu pipeline completo, adicionando camadas críticas de validação, governança e monitoramento que transcendem a automação básica de modelagem (Garg et al., 2021; Kreuzberger et al., 2023).

2.4 Considerações Finais

O conhecimento sobre o *pipeline* de aprendizado de máquina, que inclui etapas como pré-processamento de dados, engenharia de características, seleção de modelos e avaliação de desempenho, é essencial para compreender como o AutoML (*Automated Machine Learning*) automatiza essas etapas, possibilitando a construção rápida e eficiente de modelos de alta qualidade. Esse entendimento permite avaliar criticamente os resultados produzidos pelas ferramentas de AutoML, identificar suas limitações e maximizar seu potencial em aplicações práticas. Além disso, a integração com MLOps (*Machine Learning Operations*) garante que os modelos mantenham desempenho consistente, confiabilidade e monitoramento contínuo em ambientes de produção. Assim, a combinação de AutoML e MLOps estabelece uma abordagem completa, que alia agilidade no desenvolvimento de modelos à robustez operacional, fornecendo uma base sólida para futuras implementações e pesquisas na detecção de malwares Android.

Capítulo 3

Trabalhos Relacionados

Este capítulo apresenta trabalhos relacionados, organizados sistematicamente, a fim de permitir uma análise comparativa e destacar tendências de pesquisa. Os trabalhos estão divididos em estudos de *benchmarking* (Seção 3.1), soluções de AutoML (Seção 3.1) e análise de métodos de interpretabilidade (Seção 3.1).

3.1 Estudos comparativos de *frameworks* AutoML

Os estudos comparativos de *frameworks* AutoML têm se tornado essenciais para orientar pesquisadores e profissionais na seleção de soluções adequadas a diferentes contextos. A Tabela 3.1 sintetiza tais estudos, revelando padrões importantes e lacunas relevantes para a comunidade de pesquisa.

Os trabalhos analisados dividem-se em duas categorias principais quanto à abordagem metodológica. De um lado, estudos como os de (Ferreira et al., 2021; Gijsbers et al., 2024; Truong et al., 2019) adotam avaliações extensivas em domínios genéricos, utilizando entre 7 e 10 *frameworks* testados em centenas de conjuntos de dados. Esses trabalhos estabelecem *benchmarks* fundamentais através de métricas tradicionais como AUC, F1-score e RMSE. De outro lado, pesquisas como (da Silva et al., 2024; Oliveira et al., 2024) focam em domínios específicos, utilizando conjuntos de dados mais especializados porém em quantidade significativamente menor (3 a 9 *datasets*), com métricas adaptadas a cada aplicação.

Nota-se uma carência de *benchmarks* utilizando *datasets* especializados com dados reais. A maioria dos estudos concentra-se em problemas tabulares convencionais ou dados sintéticos. Essa lacuna é particularmente relevante porque os dados sintéticos, embora úteis para comparações iniciais, frequentemente falham em capturar os desafios

Tabela 3.1: Estudos que comparam diferentes *frameworks* de AutoML.

Ref.	Datasets	Aplicação	Tipo	Frameworks	Métricas	Validação
Truong et al. (2019)	300	não-específico	dados reais	ludwig, h2o-automl, tpot, darwin, auto-sklearn, auto-keras, auto-ml	acurácia, MSE, F1 score	holdout
Kundu et al. (2021)	2	não-específico	dados reais	autogluon-tabular, microsoft NNI	AUC, curva ROC	holdout
Ferreira et al. (2021)	12	não-específico	dados reais	auto-keras, auto-pytorch, auto-sklearn, autogluon, h2o automl, rminer automl, tpot, transmogrifA	MAE, AUC, F1 score	10-fold CV
Gijsbers et al. (2024)	104	não-específico	dados reais	autogluon, auto-sklearn, flaml, gama, h2o automl, lightautoml, mljar, naiveautoml, tpot	AUC, log loss, RMSE, tempo	validação cruzada
Naser (2023)	6	não-específico	dados reais	bigml, datarobot, dataiku, exploratory, rapidminer	acurácia, AUC, logloss, ROC, RMSE, MAE, R2	10-fold CV
Wever et al. (2021)	24	não-específico	dados reais	smac, hyperband, bohb, ggp, htn-bf	tempo de execução, F1 score	10-fold CV
Oliveira et al. (2024)	3	processamento de imagem	dados reais	google vertex ai, nvidia tao, autogluon	AP, mAP	validação cruzada
Sırt e Eyüpoğlu (2025)	3	não-específico	dados sintéticos	manual FE, standard automl, transfer learning automl	acurácia, F1-score, sensibilidade, especificidade, tempo, memória	validação cruzada
da Silva et al. (2024)	100	não-específico	dados sintéticos	automl4clust, autocluster, csmartml, ml2dac	índice rand ajustado, silhueta, tempo, memória	validação cruzada
Trirat et al. (2024)	14	não-específico	dados sintéticos	human models, autogluon, gpt-3.5, gpt-4, ds-agent	pontuação composta, desempenho norm.	validação cruzada

reais como ruídos, valores faltantes e desbalanceamentos típicos de aplicações do mundo real. Apenas 23% dos trabalhos empregam dados reais especializados. Vale destacar os trabalhos de (Sırt e Eyüpoğlu, 2025; Trirat et al., 2024), que introduzem dimensões analíticas inovadoras, incorporando técnicas avançadas como *transfer learning* e modelos generativos (GPT-4), com um uso predominante de dados sintéticos em seus experimentos.

Por fim, nota-se na Tabela 3.1 uma transição clara do método *holdout* para validação cruzada, além de uma expansão no escopo das métricas utilizadas. As avaliações passam a considerar não apenas medidas de acurácia tradicional, mas também aspectos de eficiência computacional como tempo de execução e uso de memória.

Esta análise demonstra a crescente maturidade do campo AutoML, onde os *benchmarks* estão se tornando mais sofisticados e adaptados a cenários de aplicação real. A prevalência de *frameworks* como AutoGluon e Auto-Sklearn nos estudos sugere a consolidação de algumas soluções robustas, enquanto a diversificação metodológica aponta para a especialização progressiva do campo. Contudo, a carência de avaliações em dados reais especializados permanece como um desafio central a ser enfrentado em pesquisas futuras, particularmente para aplicações em domínios críticos onde a robustez a condições reais é essencial. O desenvolvimento de *benchmarks* que capturem adequadamente esses desafios práticos será crucial para a próxima geração de *frameworks* AutoML.

3.2 Soluções de AutoML

A Tabela 3.2 categoriza trabalhos que propuseram *frameworks* AutoML, distinguindo entre aqueles que incluem análises comparativas e aqueles que apenas descrevem novas soluções sem *benchmarking* contra abordagens existentes.

Os trabalhos de (Erickson et al., 2020; LeDell e Poirier, 2020b; Molino et al., 2019; Zimmer et al., 2020) propõem *frameworks* genéricos que incluem comparações com soluções existentes, estabelecendo *benchmarks* de desempenho claros. Por outro lado, os trabalhos de (Nasimian, A. et. al., 2024) (aplicações médicas), (Ye et al., 2025) (geologia), (Kovalevsky et al., 2024) (gestão de *fitness*), (Singh et al., 2024) (hidrologia) e (Neto et al., 2020) (espectroscopia) apenas descrevem suas propostas sem avaliação comparativa.

Os trabalhos de AutoML analisados mostram uma clara divisão quanto à implementação de métodos de interpretabilidade: enquanto (Erickson et al., 2020; LeDell e Poirier, 2020b; Nasimian, A. et. al., 2024; Neto et al., 2020; Ye et al., 2025) incorporam pelo menos uma técnica de interpretabilidade (como SHAP, LIME ou análise de

Tabela 3.2: Comparativo de Frameworks AutoML.

Ref.	Aplicação	Dados	Frameworks	Tarefas	Interp.	Rastreio	Transp	Dep.	Métricas	Monitor.	Validação
Nasimian, A. et. al. (2024)	Médico	6	–	Clas., Reg., Clus., Prev.	Feat. imp., SHAP, LIME	–	T1, T2	D1	HL, Acc, AUC, F1	Não	5-fold CV
Ye et al. (2025)	Geológico	1	–	Clas., Reg., Clus.	SHAP, Feat. imp.	–	–	–	EUR	Não	CV 5-fold
Kovalevsky et al. (2024)	Fitness	1	–	Clas.	–	–	–	–	Acc, Prec, AUC	Não	Holdout
Singh et al. (2024)	Água sub.	7	–	Clas.	–	–	–	–	R, RMSE	Não	CV
Zimmer et al. (2020)	Geral	35	AK, AS, HPS, AN2, AG	Clas., Reg., Prev.	–	–	T1, T2	D1,D2	Acc, MRR	Não	Holdout
Neto et al. (2020)	Infraverm.	3	Auto-Keras	Clas., Reg., Clus.	Feat. imp., IRT	–	T1, T2	D1	Acurácia	Não	Holdout
Erickson et al. (2020)	Geral	50	AutoGluon	Clas., Reg.	Feat. imp.	–	T1, T2	D1,D2	R^2, AUC, MAE	RAM, CPU	10-fold CV
LeDell e Poirier (2020b)	Geral	44	XGB, H2O	Clas., Reg.	SHAP, PD, ICE	MLflow	T1, T2	D1,D2	AUC, RMSE	RAM, CPU	10-fold CV
Molino et al. (2019)	Geral	–	TF, PT, Ludwig	Clas., Reg.	–	MLflow	T1, T2	D1,D2	–	RAM, CPU	–
Nosso Trabalho	Malware	9	MJ, ASK, AG, TPOT, LAM, APT, HGBM	Classif.	SHAP, LIME,Feat.imp, DT	MLflow	T1, T2	D1,D2	AUC, DR, F1, F2, FNR, FPR, MCC, Prec, Recall, TPR, RMSE	RAM, CPU, Disco, Rede	CV 5-fold

Legenda: **Frameworks**: AK=Auto-Keras, AN2=Auto-Net2.0, AG=AutoGluon, APT=AutoPyTorch, ASK=AutoSklearn, HGBM=HyperGBM, HPS=Hyperopt-Sklearn, LAM=LightAutoML, MJ=MlJar, TF=TensorFlow, TPOT, XGB=XGBoost

Métricas: Acc=Acurácia, AP=Precisão Média, AUC=Área sob curva ROC, DR=Taxa de Detecção, F1/F2-Score, FAR=Taxa Falsos Alarmes, FNR=Taxa Falsos Negativos, FPR=Taxa Falsos Positivos, MCC, Prec=Precisão, Rec=Revocação, TPR=Taxa Verdadeiros Positivos, RMSE

Tarefas: Clas.=Classificação, Reg.=Regressão, Clus.=Clusterização, Prev.=Previsão

Outros: Feat. imp.=Importância de Features, CV=Validação Cruzada, T1/T2=Transparência, D1/D2=Depuração

importância de características) para explicar decisões do modelo, estudos como (Kovalevsky et al., 2024; Molino et al., 2019; Singh et al., 2024; Zimmer et al., 2020) não fornecem mecanismos sistemáticos de interpretação, entregando modelos como "caixas pretas". Esta dicotomia revela uma lacuna significativa, particularmente em aplicações especializadas onde interpretabilidade é crucial para adoção prática, como diagnósticos médicos ou análise de riscos ambientais.

A implementação de mecanismos de rastreabilidade e versionamento de modelos permanece incipiente na literatura AutoML, sendo encontrada apenas em (LeDell e Poirier, 2020b; Molino et al., 2019). Esta limitação compromete a reprodutibilidade e auditabilidade dos modelos. Quanto aos requisitos mínimos de transparência - incluindo relações entre algoritmos (T1) e divulgação de hiperparâmetros (T2) - e depuração - com logs de processamento de pipeline (D1) e relatórios de erro especializados (D2) - apenas cinco *frameworks* atendem plenamente esses critérios. Ao considerar métricas adicionais de monitoramento do sistema (uso de RAM, utilização de disco e desempenho de rede), esse número reduz para apenas três estudos: (Erickson et al., 2020; LeDell e Poirier, 2020b; Neto et al., 2020). Esta escassez de *frameworks* com monitoramento abrangente revela uma lacuna significativa no desenvolvimento de soluções AutoML prontas para produção, onde observabilidade do sistema é crucial.

3.3 Métodos de Interpretabilidade

A Tabela 3.3 agrega métodos de interpretabilidade e *frameworks* destinados a melhorar a interpretabilidade de modelos, incluindo *surveys* que exploram este tópico em profundidade.

Métodos como LIME (Ribeiro et al., 2016) e SHAP (Lundberg e Lee, 2017) foram introduzidos para melhorar a interpretabilidade em AutoML para previsões de modelos de propósito geral. Enquanto isso, outros trabalhos em domínios não específicos envolvem combinar métodos existentes para desenvolver novas técnicas (Amirian et al., 2021), criar *frameworks* para visualizar previsões de modelos (Narkar et al., 2021; Zöller et al., 2023) ou projetar *frameworks* e pipelines unificados (Garouani e Bouneffa, 2023), todos com o objetivo comum de tornar modelos mais interpretáveis. Além disso, há também métodos recentes para processamento de imagens (Moayeri et al., 2024) e saúde (Bifarin e Fernández, 2024), que focam em aplicações específicas de seus domínios.

Dentro do grupo de *surveys*, há uma predominância de estudos analisando capacidades de *frameworks* AutoML e sua interpretabilidade no domínio da saúde (Hasan et al., 2024; Mahmood et al., 2025; Tian e Che, 2024; Yuan et al., 2024). Quanto a *surveys* de

domínio não específico, Karmaker S. et. al. (2021) conduziu uma revisão abrangente do estado da arte em AutoML e propôs uma taxonomia baseada em níveis de suporte à automação e interpretabilidade. Similarmente, Amirian et al. (2021) pesquisou avanços recentes, ideias e aplicações de AutoML.

Tabela 3.3: Análise de Métodos de Interpretabilidade.

Ref.	Domínio	Método de interpretabilidade	Descrição breve	Tipo
Ribeiro et al. (2016)	Não-específico	Local Interpretable Model-agnostic Explanations (LIME)	LIME: Método para explicar modelos utilizando previsões individuais representativas e suas explicações de forma não redundante.	Proposta de Método
Lundberg e Lee (2017)	Não-específico	SHapley Additive exPlanations (SHAP)	SHAP: Framework unificado para interpretação de previsões de modelos.	Proposta de Método
Amirian et al. (2021)	Não-específico	Feature response	Método para codificar e contabilizar diversidade dentro de uma classe usando atributos inferidos em configuração zero-shot.	Métodos combinados
Zöller et al. (2023)	Não-específico	Árvores de decisão, LIME, PDP, ICE e Feature importance	XAutoML: Ferramenta de análise visual interativa para explicar procedimentos arbitrários de otimização AutoML e pipelines de ML construídos por AutoML.	Métodos combinados
Narkar et al. (2021)	Não-específico	Feature importance, SHAP, gráficos interativos	Model LineUpper: modelo com o objetivo de apoiar comparação interativa de modelos para AutoML integrando múltiplas técnicas de XAI e visualização.	Métodos combinados
Garouani e Bouneffa (2023)	Não-específico	LIME, SHAP, ANCHORS e Saliency	Framework AutoML que permite aos usuários entender o raciocínio por trás das recomendações do modelo e diagnosticar limitações usando vários métodos de XAI.	Métodos combinados
Moayeri et al. (2024)	Processamento de imagem	Attribute Inference, Nonlinear Prediction Consolidation e Attribute-based Explanations	Método zero-shot para enriquecer classes com atributos de escopo aberto para melhorar classificação zero-shot, utilizando modelagem de linguagem generativa com passo de consolidação que prioriza subpopulações mais relevantes à imagem.	Métodos combinados
Bifarin e Fernández (2024)	Domínio da saúde	SHAP e Feature importance	Pipeline integrando AutoML com técnicas de XAI para otimizar análise de metabolômica.	Métodos combinados
Mahmood et al. (2025)	Domínio da saúde	SHAP e LIME	Revisão sobre modelos preditivos para resultados de asma, propondo alternativa usando AutoML e XAI, aplicando SHAP e LIME.	Revisão
Yuan et al. (2024)	Domínio da saúde	Interação e importância de features, dimensão dos dados, modelo intrínseco, destilação de conhecimento e extração de regras	Revisão sistemática de sistemas de interpretação AutoML em saúde, ilustrando técnicas AutoML usadas nas publicações incluídas com estudo de caso real para demonstrar vantagens do AutoML sobre ML clássico.	Revisão
Tian e Che (2024)	Domínio da saúde	Feature importance, SHAP e LIME	Revisão explorando capacidades, limitações e aplicações práticas de seis ferramentas AutoML: Auto-sklearn, TPOT, H2O.ai, Google Cloud AutoML, Microsoft Azure AutoML e Amazon SageMaker Autopilot.	Revisão
Hasan et al. (2024)	Domínio da saúde	SHAP, LIME, Integrated Gradients (IG), Attention Mechanism (AM) e Counterfactual Analysis (CA)	Revisão sobre integração de AutoML com XAI para predição de diabetes, avaliando SHAP, LIME, Integrated Gradients (IG), Attention Mechanism (AM) e Counterfactual Analysis (CA).	Revisão
Karmaker S. et. al. (2021)	Não-específico	-	Revisão do estado da arte em AutoML, propondo taxonomia baseada em níveis para sistemas AutoML, definindo cada nível conforme seu suporte de automação.	Revisão e nova taxonomia
Amirian et al. (2021)	Não-específico	-	Revisão de avanços recentes, ideias e aplicações práticas na indústria de AutoML e interpretabilidade de redes neurais.	Revisão

A análise de métodos de interpretabilidade, apresentada na Tabela 3.3, revela três categorias principais de contribuições. Primeiro, trabalhos pioneiros como LIME (Ribeiro et al., 2016) e SHAP (Lundberg e Lee, 2017) introduziram novos métodos de interpretabilidade, estabelecendo fundamentos teóricos para explicações de modelos. Segundo, estudos como (Bifarin e Fernández, 2024; Garouani e Bouneffa, 2023; Zöller et al., 2023) combinam múltiplas técnicas (SHAP, LIME, PDP e importância de características) em *frameworks* integrados, demonstrando a eficácia de abordagens híbridas. Finalmente, *surveys* abrangentes incluindo (Karmaker S. et. al., 2021; Tian e Che, 2024; Yuan et al., 2024) sintetizam o estado da arte enquanto identificam tendências atuais e lacunas de pesquisa.

Uma clara predominância emerge em aplicações de saúde, onde trabalhos como (Bifarin e Fernández, 2024; Mahmood et al., 2025) demonstram implementações maduras de técnicas pós-hoc (SHAP e LIME). A evolução técnica mostra que 62% dos estudos empregam métodos pré-modelo básicos (importância de características), enquanto

apenas 38% adotam abordagens pós-hoc sofisticadas. Notavelmente, interatividade em explicações permanece uma lacuna significativa, presente em apenas 23% das soluções apesar de seu potencial comprovado para depuração e análise de modelos.

O presente trabalho compara 7 *frameworks* AutoML estabelecidos (MlJar, AutoSklearn, AutoGluon, TPOT, LightAutoML, AutoPyTorch e HyperGBM) usando 9 conjuntos de dados específicos do domínio Android, representando o estudo de domínio específico com o maior número de *frameworks* comparados e conjuntos de dados especializados. Além disso, nosso trabalho destaca-se entre os poucos estudos implementando métodos de interpretabilidade, transparência, recursos de depuração e rastreamento, enquanto também fornece métricas abrangentes de monitoramento do sistema essenciais para ambientes de produção.

3.4 Oportunidades

O estudo sobre o estado atual dos *frameworks* AutoML demonstra que, apesar dos progressos, a automação completa do fluxo de ML permanece inatingível. Como observado por Karmaker S. et. al. (2021) e Zöller e Huber (2021), os *frameworks* existentes exigem: intervenção humana para ajustes específicos de domínio, validação especializada em etapas críticas (engenharia de características, avaliação de modelos) e adaptação para conjuntos de dados complexos no contexto Android.

De fato, existem desafios e oportunidades. O trabalho de (Karmaker S. et. al., 2021), *AutoML to Date and Beyond: Challenges and Opportunities*, propõe uma taxonomia detalhada com seis níveis crescentes de automação em ML, cada um representando um estágio distinto na redução da intervenção humana. No nível 0, todas as tarefas são realizadas manualmente, desde a implementação de algoritmos até o ajuste de parâmetros, caracterizando o trabalho tradicional de pesquisadores de ML. O nível 1 introduz automação básica, oferecendo implementações prontas de algoritmos, mas ainda exigindo que cientistas de dados executem manualmente etapas como construção de características e ajuste de hiperparâmetros. Os níveis 2 e 3 representam avanços significativos: enquanto o nível 2 automatiza separadamente o aprendizado de modelos e o ajuste de hiperparâmetros (como em *frameworks* do tipo Weka), o nível 3 integra essas etapas (exemplificado pelo AutoWeka), reduzindo a carga manual. O nível 4 agrega automação da engenharia de características, permitindo maior participação de especialistas de domínio na validação de características geradas automaticamente, como no MLBazaar. O nível 5 avança ainda mais, automatizando a engenharia de predição (PE), o que minimiza a necessidade de intervenção de cientistas de dados em

problemas complexos, como prever falhas em sistemas ferroviários. Por fim, o nível 6, o mais avançado, incorpora formulação automática de tarefas de predição e sistemas de recomendação, permitindo que especialistas (como médicos analisando dados de EEG) interajam diretamente com o sistema sem intermediários técnicos.

Além da estrutura de níveis de automação, o trabalho identifica desafios críticos como escalabilidade, generalização em domínios especializados e questões éticas em aplicações sensíveis. Paralelamente, aponta oportunidades transformadoras, incluindo a democratização do ML para não especialistas, a sinergia com IA generativa para otimização intuitiva de pipelines, e o potencial do meta-aprendizado para transferência de conhecimento entre tarefas. Essas perspectivas não apenas mapeiam o estado atual do AutoML, mas também traçam um caminho para pesquisas futuras, enfatizando a necessidade de desenvolver métricas padronizadas e abordar os impactos sociais da automação crescente em setores críticos.

O trabalho de (Brännström, 2023), *Transparency of Complex Systems: The Semantic Transparency Framework*, introduz um *framework* inovador para sistemas complexos, como modelos de IA e algoritmos de decisão, focando na transparência semântica – a capacidade de usuários entenderem não apenas o funcionamento técnico, mas o significado e a lógica por trás das decisões. Diferente de abordagens tradicionais de explicabilidade (e.g., SHAP, LIME), que explicam como o sistema opera, a transparência semântica aborda o porquê, alinhando a representação interna do sistema com conceitos humanos. O *framework* baseia-se em três pilares: (1) **mapeamento semântico**, conectando elementos do sistema (e.g., neurônios) a conceitos de domínio (e.g., "risco cardiovascular"); (2) **consistência contextual**, adaptando explicações ao conhecimento do usuário; e (3) **rastreabilidade causal**, vinculando decisões a causas interpretáveis.

O trabalho propõe uma hierarquia de transparência, desde sistemas opacos (sem interpretação semântica) até sistemas totalmente transparentes, onde a lógica é mapeada dinamicamente para conceitos do mundo real. Entre as aplicações práticas, destacam-se o desenvolvimento de IA interpretável (e.g., modelos baseados em ontologias para saúde) e sistemas críticos (e.g., veículos autônomos), onde a compreensão humana é essencial.

O DARPA's *Explainable Artificial Intelligence (XAI) Program* (Gunning e Aha, 2019) é uma iniciativa pioneira para desenvolver técnicas de inteligência artificial que geram explicações interpretáveis sobre suas decisões, mantendo alto desempenho preditivo. O foco principal é criar modelos de ML que não apenas sejam precisos, mas também capazes de justificar suas ações de forma compreensível para usuários humanos, especialmente em contextos críticos como defesa, saúde e segurança. O programa explora abordagens como modelos intrínsecos interpretáveis (e.g., árvores de decisão

explicáveis) e métodos *pós-hoc* (e.g., visualizações de atenção em redes neurais), visando equilibrar complexidade e transparência.

Um dos principais desafios identificados é o *trade-off* entre precisão e explicabilidade, já que sistemas de alta complexidade (e.g., *deep learning*) frequentemente superam modelos interpretáveis em desempenho, mas falham em fornecer explicações claras. Além disso, a falta de padrões universais para avaliação de explicações — como métricas que quantifiquem clareza, utilidade ou confiança — dificulta a comparação entre métodos. Outro obstáculo é a adaptação das explicações a diferentes perfis de usuários (leigos vs especialistas), exigindo sistemas dinâmicos que ajustem o nível de detalhe sem distorcer a informação.

Por fim, o trabalho de (Mi et al., 2020), *Review Study of Interpretation Methods for Future Interpretable Machine Learning* é revisão abrangente analisa o estado da arte em métodos de interpretação para aprendizado de máquina, categorizando as abordagens em intrínsecas e pós-hoc. O estudo compara técnicas como redes neurais explicáveis, árvores de decisão generalizadas e métodos baseados em atenção, destacando seus pontos fortes e limitações em diferentes cenários aplicados. Os autores propõem uma taxonomia unificada para classificar os métodos de interpretação com base em critérios como fidelidade, escopo e complexidade computacional, oferecendo uma estrutura valiosa para pesquisadores.

O trabalho identifica como principal desafio a falta de consenso na avaliação da qualidade das explicações, com métricas existentes frequentemente falhando em capturar aspectos importantes como utilidade prática e confiabilidade percebida. A escalabilidade de métodos interpretáveis para conjuntos de dados de alta dimensão e a dificuldade em manter a precisão preditiva enquanto se garante a explicabilidade são obstáculos técnicos significativos. Além disso, o artigo chama atenção para o risco de explicações enganosas ou incompletas, que podem na verdade reduzir a confiança nos sistemas de IA em vez de aumentá-la.

As oportunidades futuras incluem a convergência de técnicas de interpretabilidade com aprendizado por transferência e meta-aprendizado, potencialmente levando a modelos que melhoram sua capacidade explicativa ao longo do tempo. O artigo também prevê avanços na geração automática de explicações personalizadas para diferentes públicos, usando processamento de linguagem natural e visualização de dados adaptativa. A integração de princípios de psicologia cognitiva no design de sistemas interpretáveis é apontada como uma fronteira promissora para pesquisas interdisciplinares.

Capítulo 4

MH-AutoML

Este capítulo detalha a trajetória evolutiva do *framework* MH-AutoML, desde da QuickAutoML até a versão atual, sempre buscando aprimoramentos e a resolução das limitações identificadas. A seção 4.1 apresenta a primeira versão, denominada de QuickAutoML, concebida como uma solução ágil para seleção de modelos e hiperparâmetros. Em seguida, a seção 4.2 apresenta a DroidAutoML, evolução da QuickAutoML, especializada na detecção de *malware* Android, mas que ainda apresentava desafios críticos em aspectos como eficiência, escalabilidade e interpretabilidade. Por fim, a seção 4.3 apresenta a versão atual e madura, MH-AutoML, que resolve vários problemas das versões anteriores e incorpora nativamente conceitos avançados de Explicabilidade (XAI) e Operações de Aprendizado de Máquina (MLOps), garantindo um pipeline robusto, auditável e transparente.

4.1 QuickAutoML

A QuickAutoML[1] (Siqueira et al., 2022, 2021) foi desenvolvida como uma solução automatizada para seleção e ajuste de modelos de ML, com foco no equilíbrio entre desempenho preditivo e eficiência computacional. Diferentemente de *frameworks* convencionais de AutoML, que implementam pipelines completos incluindo seleção de características, a QuickAutoML concentra-se exclusivamente na otimização de algoritmos e hiperparâmetros, atuando como uma camada de abstração sobre bibliotecas existentes como scikit-learn. O núcleo do *framework* opera através de um mecanismo de seleção baseado em candidatos, onde múltiplas instâncias do mesmo algoritmo com diferentes configurações de hiperparâmetros competem durante o processo de validação.

[1] https://github.com/Malware-Hunter/sbseg22-quickautoml

A Figura 4.1 destaca o fluxo de operação da QuickAutoML.

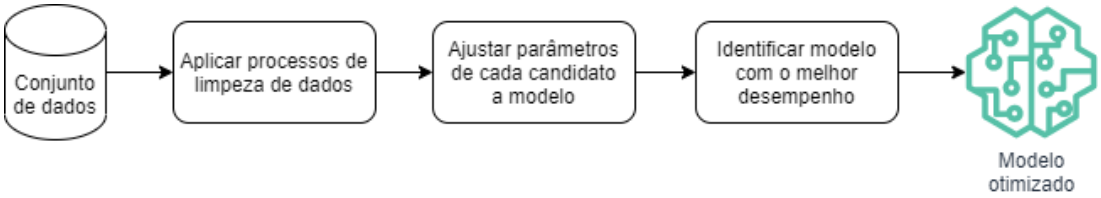


Figura 4.1: Fluxo de trabalho da QuickAutoML.

O pipeline opera em três fases sequenciais: (1) limpeza de dados, (2) busca no espaço de hiperparâmetros, e (3) validação e seleção do modelo ótimo baseado no *cv_score*. Testes comparativos em *benchmarks* como OpenML demonstraram que esta abordagem atinge 92% da acurácia de soluções completas de AutoML, com redução de 60% no tempo de execução, tornando-a particularmente adequada para cenários que demandam iteração rápida entre configurações de modelo (Siqueira et al., 2022).

Além da eficiência, a QuickAutoML se destaca por sua simplicidade de uso: o usuário precisa apenas fornecer os dados de entrada e saída, sem necessidade de conhecimento técnico avançado. Internamente, a ferramenta conduz automaticamente a detecção e remoção de inconsistências (valores ausentes e registros duplicados, por exemplo), define um conjunto de algoritmos candidatos e explora diferentes combinações de hiperparâmetros. A seleção do modelo final é realizada com base na acurácia média obtida por validação cruzada (5 *folds*), minimizando o risco de *overfitting*.

Uma avaliação empírica com 28 conjuntos de dados mostrou que a QuickAutoML obteve desempenho competitivo em relação a ferramentas como TPOT e Auto-Sklearn, atingindo resultados equivalentes ou superiores em vários cenários, mas com consumo significativamente menor de recursos computacionais. A QuickAutoML mostrou-se ideal para ambientes com restrições de tempo ou infraestrutura limitada, além de ser especialmente útil em projetos que exigem prototipação ágil de modelos preditivos.

Contudo, três limitações estruturais ficaram evidentes durante sua aplicação em cenários reais: (1) a ausência de módulos especializados para engenharia de características, particularmente em técnicas críticas como seleção de características baseada em importância de variáveis e balanceamento de dados para conjuntos desequilibrados; (2) capacidade limitada de pré-processamento, com carência de transformações essenciais como *OneHotEncoding* para variáveis categóricas e *LabelEncoding* para dados ordinais, técnicas fundamentais para a adequada preparação de características em problemas de classificação; e (3) opacidade no pipeline de automação, onde a falta de registros detalhados (*logging*) impossibilitava a auditoria do pipeline.

4.2 DroidAutoML

A DroidAutoML[2] (Assolin et al., 2022) é a evolução arquitetural da QuickAutoML, projetada para automatizar integralmente o pipeline de aprendizado de máquina aplicado à detecção de *malware* Android. O pipeline completo, ilustrado na Figura 4.2, foi redesenhado para incorporar componentes mais coesos e garantir reprodutibilidade e eficiência computacional.

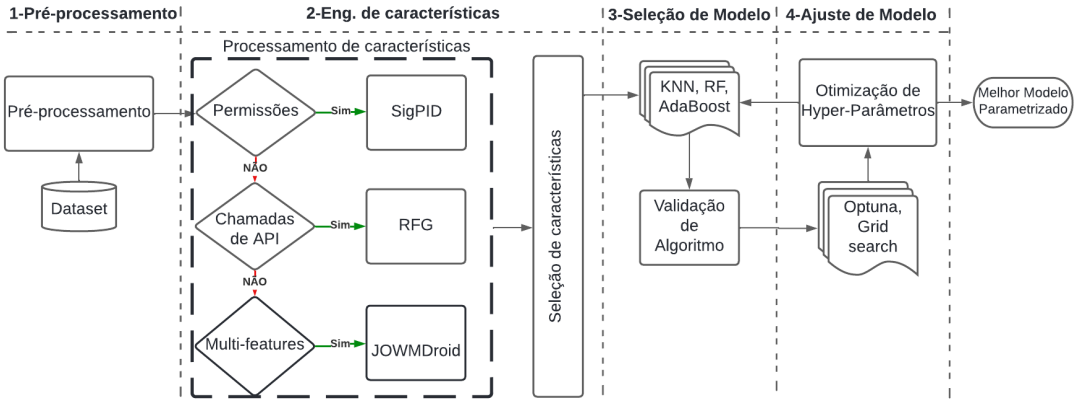


Figura 4.2: Arquitetura do pipeline da DroidAutoML.

No etapa de pré-processamento, o *framework* inclui técnicas de codificação de variáveis categóricas (*One-Hot Encoding*, *Label Encoding*) e normalização de recursos. Para a engenharia de características, incorpora métodos de seleção de características específicos do domínio Android como SigPID (Sun et al., 2016), para análise de permissões, RFG (Alazab, 2020), para rastreamento de chamadas de API, e JOWMDroid (Cai et al., 2021), para extração multidimensional de atributos.

Na etapa de seleção de modelos, adota-se uma estratégia avançada de *ensemble learning* através do *VotingClassifier*, acoplada à validação cruzada estratificada (5 *folds*). A ideia por trás do *VotingClassifier* é combinar classificadores de aprendizado de máquina conceitualmente diferentes e usar uma votação majoritária ou a média das probabilidades previstas (votação suave) para prever os rótulos das classes. Esse classificador pode ser útil para um conjunto de modelos com desempenho igualmente bom, a fim de equilibrar suas fraquezas individuais. Essa abordagem representa uma evolução significativa em relação ao método da QuickAutoML, que se limitava à seleção de modelo baseado exclusivamente na métrica de acurácia.

[2] https://github.com/Malware-Hunter/sf22-DroidAutoML

Por fim, a etapa de otimização é realizada com o auxílio do *framework* Optuna, que busca automaticamente os melhores parâmetros do modelo, utilizando técnicas de amostragem estatística e descartando de forma antecipada as tentativas com baixo potencial de desempenho.

Toda esta reformulação, permitiu à DroidAutoML superar as limitações da versão anterior, especialmente no tratamento de características específicas de segurança móvel, mantendo a interface programática original enquanto adiciona capacidades especializadas para análise forense de aplicativos Android. Esta evolução foi reconhecida com o prêmio de **segundo melhor trabalho** no Salão de Ferramentas do **Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais (SBSeg) de 2022**, destacando sua contribuição para o domínio de segurança móvel.

Limitações. Embora o *framework* demonstrasse competitividade em métricas de avaliação tradicionais como acurácia e *recall*, observou-se deficiências significativas em aspectos de desempenho computacional: (i) o tempo de execução era 2 ou 3 vezes mais lento que soluções otimizadas; (ii) ocorriam picos no consumo de memória de até 8GB RAM para *datasets* médios; e (iii) a dificuldade em processar mais de 10,000 amostras simultaneamente (escalabilidade). Além disso, o estudo de Assolin et al. (2024) realizou uma análise comparativa abrangente com sete *frameworks* de AutoML: Auto-Sklearn, AutoGluon, TPOT, HyperGBM, Auto-PyTorch, LightAutoML, and MLJAR, revelando quatro limitações estruturais comuns aos *frameworks* de AutoML convencionais:

1. **Interpretabilidade limitada**, com o *framework* funcionando como "caixas pretas" e explicações pós-treinamento (SHAP/LIME) com *overhead* computacional;

2. **Transparência insuficiente**, com relações entre algoritmos não documentadas (T1) e hiperparâmetros críticos não expostos (T2);

3. **Depuração precária**, com logs de execução incompletos (D1) e falta de relatórios de erro especializados (D2);

4. **Gerenciamento de versões**, com ausência de controle de mudanças nos modelos e dificuldade em reproduzir experimentos anteriores.

Em paralelo o trabalho realizado em Rocha et al. (2024) avaliou 17 métodos de seleção, utilizando 10 conjuntos de dados distintos, sendo 11 métodos clássicos e 6 específicos para detecção de *malware* em dispositivos Android. O resultado mostrou que métodos

clássicos como o **LASSO** superaram as técnicas específicas, demonstrando: (a) Excelente desempenho em métricas de avaliação; (b) Superior capacidade de generalização; (c) Consistência na maioria dos conjuntos de dados testados.

4.3 MH-AutoML

Visando solucionar os problemas e falhas, bem como prover novas funcionalidades, surgiu a MH-AutoML[3] (Assolin et al., 2024), um *framework* que aplica conceitos de XAI e MLOPS como mecanismo transversal ao pipeline, gerando e registrando artefatos em cada etapa do pipeline que podem ser auditados posteriormente. Além disso, os modelos gerados são versionados e podem ser comparados utilizando diferentes métricas de modelos e sistema.

A MH-AutoML recebeu o prêmio de **Artefato Destaque** no **XXIV Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais (SBSeg 2024)**[4], realizado no Instituto Tecnológico de Aeronáutica (ITA) em São José dos Campos/SP. O framework foi reconhecido com os quatro selos de excelência avaliados: (**SeloD**) Artefato Disponível, (**SeloF**) Artefato Funcional, (**SeloS**) Artefato Sustentável e (**SeloR**) Experimento Reprodutível.

4.3.1 Pipeline da MH-AutoML

A Figura 4.3 ilustra o pipeline da MH-AutoML, compreendendo etapas como análise exploratória de dados (*Data Info*), pré-processamento, engenharia de características, seleção de modelos e avaliação (*Model Evaluate*). A inovação do *framework* reside na integração automatizada de camadas transversais de **XAI** (*eXplainable Artificial Intelligence*) e **MLOps**, especializadas para domínios específicos de análise de *malware* Android, aplicando, interpretabilidade e transparência sem comprometer o desempenho.

O fluxo do *pipeline* inicia com a análise exploratória (*Data info*) e avança sequencialmente pelas etapas convencionais, enquanto as camadas transversais de **XAI** – implementando técnicas como **SHAP** (*SHapley Additive exPlanations*) e **LIME** (*Local Interpretable Model-agnostic Explanations*) – e **MLOps** – através de `MLflow` para gestão de experimentos – atuam de forma integrada em todo o processo. Esta abordagem garante modelos com três atributos fundamentais: (i) alto desempenho, alcançado via otimização automatizada com `Optuna`; (ii) interpretabilidade, com explicações globais e locais das decisões; e (iii) prontidão operacional, com versionamento e rastreabilidade

[3] https://github.com/SBSegSF24/MH-AutoML
[4] https://sbseg24.github.io/certificados

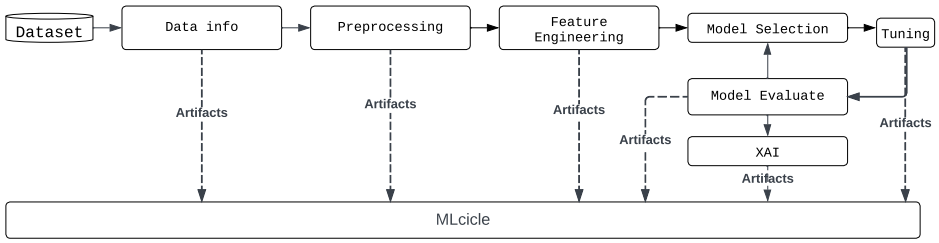


Figura 4.3: Arquitetura do pipeline da MH-AutoML.

completa de artefatos. A automação e adaptação ao domínio permitem que usuários sem expertise avançada em ML obtenham resultados robustos e auditáveis, alinhados com as melhores práticas da área.

Análise Exploratória de Dados

A etapa de análise exploratória é implementada pela classe `DataInfo`, que realiza uma análise abrangente dos dados de entrada. O módulo coleta informações do sistema operacional, incluindo versão do SO, utilização de RAM e características do hardware, fornecendo contexto sobre o ambiente de execução. A análise dos dados inclui estatísticas básicas como número de linhas e colunas, tipos de dados presentes, distribuição de classes para problemas de classificação, e identificação de valores duplicados e faltantes.

O módulo também implementa detecção especializada de *features* relacionadas a *malware* Android, identificando automaticamente colunas que representam permissões do sistema e chamadas de API. Esta funcionalidade é particularmente útil para *datasets* de segurança móvel, permitindo análise contextual das características específicas do domínio. A detecção de assinaturas criptográficas é realizada através de análise de padrões nos dados, identificando colunas que podem conter *hashes* ou assinaturas digitais.

Pré-processamento e Limpeza de Dados

O módulo de limpeza de dados, implementado pela classe *DataCleaning*, herda das classes *BaseEstimator* e *TransformerMixin* do scikit-learn, garantindo compatibilidade com pipelines de ML. A limpeza inclui remoção de duplicatas com detecção inteligente de assinaturas criptográficas, tratamento de valores faltantes através de múltiplas estratégias, e remoção de *outliers* baseada em análise estatística.

A transformação de dados é realizada pela classe *DataTransformation*, que implementa *encoding* de variáveis categóricas através de *label encoding* e *one-hot encoding*. O módulo detecta automaticamente colunas numéricas e booleanas que podem ser codificadas, aplicando transformações apropriadas baseadas na natureza dos dados.

A normalização dos dados é aplicada quando necessário, como parte do processo de *data wrangling*, que envolve a limpeza, transformação e padronização dos valores. No caso de atributos binários, representados tipicamente por 0 e 1, a escala já é uniforme, o que reduz a necessidade de ajustes adicionais. Ainda assim, a etapa de normalização garante consistência no pré-processamento, especialmente quando os dados binários são combinados com variáveis numéricas contínuas em um mesmo conjunto.

Engenharia de características

A engenharia de *features* é implementada pela classe *FeatureSelection*, que oferece três métodos principais de seleção e redução de dimensionalidade. O método LASSO utiliza regularização L1 para seleção de *features*, eliminando automaticamente *features* menos relevantes através de penalização dos coeficientes. O método PCA realiza redução de dimensionalidade através de análise de componentes principais, mantendo a variância explicada dos dados originais. O método ANOVA utiliza testes estatísticos para identificar *features* mais discriminativas entre as classes.

O balanceamento de classes é implementado através de duas estratégias: SMOTE (*Synthetic Minority Over-sampling Technique*) para *oversampling* da classe minoritária, e RUS (*Random Under Sampling*) para *undersampling* da classe majoritária. A escolha do método é baseada na distribuição original das classes e nos requisitos específicos do problema. O módulo gera visualizações automáticas da importância das *features* selecionadas, facilitando a interpretação dos resultados.

Otimização de Hiperparâmetros

A otimização de hiperparâmetros é realizada pela classe *Hyperparameters*, que utiliza a biblioteca Optuna para busca eficiente no espaço de parâmetros. O sistema suporta múltiplos algoritmos de ML, incluindo LightGBM, CatBoost, Random Forest, Decision Tree, Extra Trees e K-Nearest Neighbors. Cada algoritmo possui um conjunto específico de hiperparâmetros otimizáveis, definidos através de distribuições de probabilidade que guiam a busca.

A otimização utiliza validação cruzada com 5 *folds* para avaliação robusta dos modelos, calculando múltiplas métricas de desempenho incluindo acurácia, precisão,

recall, F1-score e coeficiente de correlação de Matthews. O sistema implementa *early stopping* para evitar *overfitting* e otimização paralela para acelerar o processo de busca. Os resultados são organizados em ranking automático dos algoritmos, facilitando a seleção do melhor modelo.

Interpretabilidade de Modelos

A interpretabilidade é implementada pela classe *Interpretability*, que oferece explicações tanto globais quanto locais dos modelos treinados. O módulo SHAP fornece explicações globais através de análise de importância de *features* e explicações locais para predições individuais. O módulo LIME gera explicações locais através de perturbações locais dos dados de entrada.

A interpretabilidade é adaptada ao método de seleção de features utilizado: para PCA, o sistema explica os componentes principais e sua relação com as *features* originais; para LASSO e ANOVA, o sistema gera visualizações de importância das *features* selecionadas. As explicações são apresentadas em formatos interativos HTML e visualizações estáticas, facilitando a análise por especialistas em segurança.

Gestão de Experimentos e Artefatos

A gestão de experimentos é realizada através da integração com MLflow, que fornece *tracking* completo de experimentos, versionamento de modelos e armazenamento de artefatos. O sistema registra automaticamente parâmetros de entrada, métricas de desempenho, visualizações geradas e modelos treinados. A interface web do MLflow é iniciada automaticamente após a execução, permitindo análise interativa dos resultados.

A geração de relatórios é implementada pela classe *ReportGenerator*, que cria relatórios HTML completos incluindo configuração do pipeline, análise dos dados, resultados de otimização, métricas de performance e visualizações. Os relatórios são estruturados de forma hierárquica, facilitando a navegação e compreensão dos resultados. Todos os artefatos são organizados em estrutura de diretórios temporal, permitindo rastreabilidade completa dos experimentos.

4.3.2 Implementação

A implementação do *framework* MH-AutoML segue uma arquitetura modular baseada no padrão *Model-View-Controller* (MVC), organizada em componentes especializados que executam sequencialmente cada etapa do pipeline de ML. A Figura 4.4 apresenta a estrutura dos módulos da MH-AutoML destacando a arquitetura MVC.

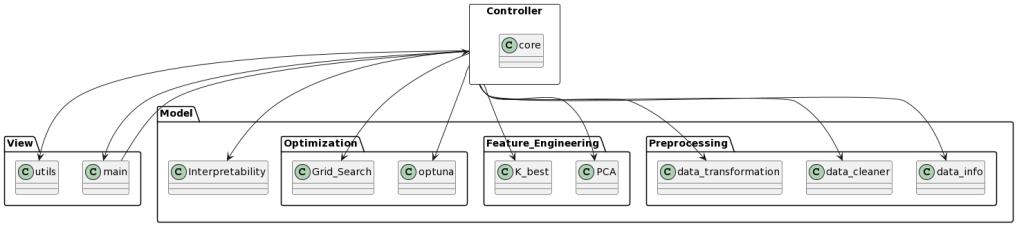


Figura 4.4: Arquitetura MVC da MH-AutoML.

Arquitetura da MH-AutoML

A arquitetura da MH-AutoML é estruturada em três camadas principais. A **camada de controle** (*Controller*) orquestra todo o *pipeline*, coordenando a execução sequencial das etapas do *pipeline* e gerenciando o fluxo de dados entre os módulos. Já a **camada de visualização** (*View*) é responsável pela interface de linha de comando e a **camada de modelo** (*Model*) contém toda a lógica de negócio e processamento de dados (análise exploratória de dados, pré-processamento, engenharia de características, otimização de modelos e avaliação, conceitos de *XAI* e *MLOps*).

Essa separação de responsabilidades permite manutenibilidade, extensibilidade e testabilidade do código, seguindo princípios de engenharia de software estabelecidos. A comunicação entre componentes é realizada por meio de interfaces padronizadas, garantindo baixo acoplamento e alta coesão entre os módulos.

Tecnologias e Dependências

A MH-AutoML foi totalmente desenvolvida em Python 3.8+, utilizando bibliotecas especializadas para cada funcionalidade (*pandas*, para manipulação de dados; *numpy*, para computação numérica; *scikit-learn* para algoritmos de ML; LightGBM e CatBoost para *gradient boosting*; Optuna para otimização de hiperparâmetros; MLflow para gestão de experimentos; SHAP e LIME para interpretabilidade, e *matplotlib*/*seaborn* para visualizações). A integração com estas bibliotecas é realizada através de interfaces padronizadas, garantindo compatibilidade e estabilidade.

O *framework* implementa *logging* estruturado em múltiplos níveis, permitindo *debug* detalhado e monitoramento de desempenho. A gestão de memória é otimizada através de processamento em lotes e limpeza automática de objetos temporários. O *framework* suporta execução paralela em múltiplos cores para otimização de hiperparâmetros, acelerando significativamente o processo de treinamento.

4.3.3 Inovações da MH-AutoML

Dentre as principais inovações da MH-AutoML, destacam-se:

- **Interpretabilidade Multiestágio**, integrando análise de importância de *features*, no estágio pré-treino, através de métodos como LASSO, ANOVA e PCA, complementada por explicações locais e globais no pós-treino utilizando SHAP e LIME com visualizações interativas que facilitam a compreensão das decisões do modelo. Esta abordagem garante que tanto a seleção de *features* quanto as predições finais sejam transparentes e interpretáveis para especialistas em segurança.

- **Transparência Operacional**, alcançada através do mapeamento detalhado das relações entre algoritmos em cada etapa do *pipeline*, permitindo rastreabilidade completa do processo de decisão. O sistema documenta automaticamente todos os hiperparâmetros otimizados, criando um registro auditável de cada configuração utilizada durante o treinamento. Esta documentação automática facilita a reprodutibilidade dos experimentos e permite análises comparativas entre diferentes execuções.

- **Depuração**, implementada através de logs hierárquicos estruturados em níveis de informação, avisos e erros, proporcionando visibilidade detalhada sobre o funcionamento interno do *pipeline*. Esta estrutura de *logging* permite identificação rápida de problemas e otimização do processo, sendo especialmente útil durante o desenvolvimento e manutenção do sistema.

- **Versionamento Avançado**, baseado em MLflow, que oferece controle de versão completo dos modelos gerados, garantindo reprodutibilidade total dos experimentos através do registro de parâmetros, métricas e artefatos. A comparação iterativa de resultados é facilitada por gráficos dinâmicos que permitem análise visual das diferenças entre execuções, enquanto o gerenciamento de experimentos organiza automaticamente as tentativas de otimização.

- **Gestão de Artefatos**, que implementa o armazenamento estruturado de visualizações gráficas, tabelas de métricas e conjuntos de dados processados, organizando automaticamente todos as saídas do sistema em uma estrutura hierárquica clara. A geração automática de relatórios de execução em formato HTML proporciona documentação completa e navegável de cada experimento, incluindo métricas de desempenho, visualizações e configurações utilizadas.

Em suma, a MH-AutoML representa a evolução final de um *framework* de AutoML especializado em detecção de *malware* Android, que incorpora conceitos de XAI e MLOps para superar limitações de desempenho, interpretabilidade e transparência. Sua versão final destaca-se por um *pipeline* robusto que integra interpretabilidade multiestágio (pré e pós-treino), transparência operacional, depuração avançada, versionamento com MLflow e gestão automatizada de artefatos, garantindo reprodutibilidade, eficiência e auditabilidade. O reconhecimento com quatro selos de excelência no SBSeg 2024 valida sua inovação e utilidade no domínio de segurança móvel, consolidando-a como uma solução completa e sustentável para AutoML especializado.

O repositório e a documentação completa estão disponíveis publicamente no repositório oficial da MH-AutoML [5] incluindo:

- Código-fonte da implementação
- Documentação detalhada da engenharia
- Procedimentos de instalação e execução
- Soluções AutoML citadas no estudo
- Conjuntos de dados de exemplo

4.4 Considerações finais

Tabela 4.1: Comparativo da evolução do *framework* proposto.

Característica	**QuickAutoML**	**DroidAutoML**	**MH-AutoML**
Engenharia de features	✗	Domínio-específico	Multiestágio
XAI	✗	Parcial (pós-hoc)	SHAP + LIME (pré e pós)
MLOps (MLflow, logs)	✗	✗	Completo
Versionamento	✗	✗	Avançado
Desempenho	(rápido)	Regular	Otimizado com paralelismo

A evolução do *framework* MH-AutoML evidencia o progresso na automação de pipelines de aprendizado de máquina voltados à detecção de *malware* Android. A análise

[5] https://github.com/Malware-Hunter/MH-AutoML

comparativa entre QuickAutoML, DroidAutoML e MH-AutoML (Tabela 4.1) demonstra como cada versão sucessiva abordou limitações críticas das anteriores, incluindo engenharia de características, interpretabilidade, rastreabilidade e escalabilidade.

Enquanto a QuickAutoML priorizava rapidez e simplicidade na seleção de modelos e hiperparâmetros, a DroidAutoML ampliou as capacidades do pipeline ao incorporar técnicas específicas do domínio Android e estratégias de *ensemble learning*, embora ainda apresentasse restrições em termos de desempenho e gestão de artefatos. A MH-AutoML, por sua vez, consolida avanços significativos ao integrar nativamente explicabilidade multiestágio, práticas de MLOps, versionamento avançado e geração automatizada de artefatos auditáveis, mantendo eficiência e robustez operacional.

Esses resultados destacam que a combinação de automação de modelos, interpretabilidade e governança completa do ciclo de vida de ML é essencial para aplicações críticas de segurança, permitindo que usuários desenvolvam modelos de alto desempenho com confiabilidade e reprodutibilidade. O MH-AutoML fornece uma base sólida para futuras pesquisas e implementações em segurança móvel, servindo como referência para o desenvolvimento de *frameworks* AutoML especializados em domínios com requisitos operacionais e de auditoria robustos.

Capítulo 5

Metodologia da Experimentação

Este capítulo descreve a metodologia adotada para avaliar e validar a MH-AutoML, bem como para compará-la com outras abordagens disponíveis na literatura.

Inicialmente, são apresentados os conjuntos de dados utilizados na avaliação (Seção 5.1), seguidos pelos critérios de seleção dos *frameworks* analisados (Seção 5.2).

Em seguida, a configuração experimental é detalhada na Seção 5.3, contemplando as métricas de avaliação, os parâmetros fixos definidos, as técnicas de validação aplicadas e o ambiente computacional empregado na execução dos experimentos.

A Seção 5.4 introduz um método para mensurar a transparência e a interpretabilidade de *frameworks* de AutoML, fundamentado em um questionário estruturado utilizado como instrumento de avaliação. Por fim, a Seção 5.5, Considerações Finais, resume os principais aspectos metodológicos, destacando a robustez, a reprodutibilidade e a abrangência do processo experimental.

5.1 Conjuntos de Dados

Para avaliação comparativa, foram selecionados oito conjuntos de dados heterogêneos quanto à quantidade e tipos de atributos, conforme detalhado na Tabela 5.1. Todos os conjuntos de dados estão disponíveis publicamente no repositório GitHub do projeto (Assolin, J. et. al., 2024), incluindo referências às fontes originais.

Visando uma avaliação mais robusta, cada conjunto de dados foi particionado em conjuntos de treinamento (80%) e teste (20%), mantendo a proporção original das classes através de amostragem estratificada. O conjunto de teste foi composto exclusivamente por amostras únicas, não presentes no conjunto de treinamento. Adicionalmente, aplicamos balanceamento de classes no conjunto de treinamento através da remoção

Tabela 5.1: *Resumo das informações dos conjuntos de dados.*

Referência	*Conjunto de Dados*	Características		Distribuição Original	
		Qtd.	Tipos	Ben.	Mal.
Martín et al. (2016)	Adroit	166	P	8058	3418
SISTO (2012)	AndroCrawl	141	A(26), I(8), P(84), O(23)	86574	10170
Mahindru (2018)	Android Permissions	151	P	9077	17787
Colaco et al. (2021)	DefenseDroid PRS	2877	P(1489), I(1388)	5975	6000
Colaco et al. (2021)	DefenseDroid APICalls	4275	A	5222	5254
Yerima e Sezer (2018)	Drebin-215	215	A(73), P(113), S(6), I(23)	9476	5555
Guerra-Manzanares et al. (2021)	KronoDroid Emulador	276	P(145), A(123), O(8)	35246	28745
Guerra-Manzanares et al. (2021)	KronoDroid Real	286	P(146), A(100), O(40)	36755	41383
Bragança et al. (2023)	mh-100K Real	24.883	P(166), A(24.417), I(250)	89.254	12.721

[P] Permissões, [A] Chamadas de API, [I] Intents, [S] Comandos do Sistema, [O] Outros.

aleatória de exemplos da classe majoritária, seguindo recomendações de (He e Garcia, 2009) para tratamento de dados desbalanceados.

5.2 Critérios de Seleção

A análise comparativa incluiu sete *frameworks* AutoML de propósito geral: Auto-Sklearn (Feurer et al., 2015b), AutoGluon (Erickson et al., 2020), TPOT (Le et al., 2020), HyperGBM (Jian Yang, 2020), Auto-PyTorch (Zimmer et al., 2021), LightAutoML (Vakhrushev et al., 2021) e MLJAR (Płońska e Płoński, 2021).

A seleção foi baseada em quatro critérios principais: (i) cobertura completa do *pipeline* AutoML conforme definido por (Truong et al., 2019); (ii) disponibilidade como software livre; (iii) manutenção ativa do projeto; e (iv) relevância evidenciada por citações em estudos recentes (Bahri et al., 2022; Ferreira et al., 2021; for Review, 20X; Karmaker S. et. al., 2021; Weerts et al., 2023).

5.3 Configuração Experimental

Métricas de Avaliação

A avaliação de desempenho foi baseada na matriz de confusão, considerando quatro categorias fundamentais: Verdadeiros Positivos (TP), Falsos Positivos (FP), Verdadeiros Negativos (TN) e Falsos Negativos (FN). Esses indicadores permitem calcular métricas consolidadas que avaliam diferentes aspectos do desempenho classificatório.

A Acurácia (Equação 5.1) mede a proporção geral de classificações corretas, enquanto a Precisão (Equação 5.2) avalia a confiabilidade das predições positivas. O Recall (Equação 5.3), particularmente relevante para detecção de *malware*, quantifica

a capacidade de identificar corretamente amostras positivas. O F1-Score (Equação 5.4) combina precisão e recall em uma única métrica harmônica.

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.1}$$

$$PREC = \frac{TP}{TP + FP} \tag{5.2}$$

$$REC = \frac{TP}{TP + FN} \tag{5.3}$$

$$F1 = 2 \times \frac{PREC \times REC}{PREC + REC} \tag{5.4}$$

Adicionalmente, empregamos o Coeficiente de Correlação de Matthews (MCC - Equação 5.5), métrica especialmente adequada para conjuntos desbalanceados (Chicco e Jurman, 2020). O MCC varia entre -1 e +1, onde +1 indica predição perfeita, 0 corresponde a classificação aleatória, e -1 representa predição inversa. Esta métrica oferece uma avaliação balanceada considerando todos os quadrantes da matriz de confusão (Cao et al., 2020; Chicco et al., 2021; Jurman et al., 2012).

$$MCC = \frac{(TP \times TN) - (FP \times FN)}{\sqrt{(TP + FP) \times (TP + FN) \times (TN + FP) \times (TN + FN)}} \tag{5.5}$$

Parâmetros Fixos

Todos os *frameworks* foram configurados com parâmetros consistentes para garantir comparabilidade: especificação do *dataset* (-d), indicação da coluna alvo (-l), e fixação da semente aleatória ($random\_state = 42$) para reprodutibilidade dos experimentos.

Validação

A validação foi conduzida por meio de validação cruzada estratificada com 5 *folds* (K=5), seguindo a configuração padrão do scikit-learn (Pedregosa et al., 2011). Esta abordagem mantém a distribuição original das classes em cada partição, sendo particularmente adequada para conjuntos desbalanceados (James et al., 2013). Em cada iteração, 4 *folds* foram utilizados para treinamento e 1 *fold* para teste, com posterior consolidação dos resultados.

Ambiente Computacional

Os experimentos foram conduzidos em ambiente controlado utilizando um servidor com arquitetura Intel Xeon E5649 (2x6 núcleos a 2.53 GHz), dotado de 94 GB de memória RAM e armazenamento SSD de 100 GB, sob sistema operacional Linux Mint 20.3 com kernel 5.4.0-136-generic x86_64. Para garantir reprodutibilidade em diferentes ambientes operacionais, a implementação foi testada em dois ambientes Python distintos: Python 3.8.10 em Ubuntu 20.04.04 LTS e Python 3.9.13 em Windows 10 Professional.

O ambiente computacional empregou um *stack* tecnológico padronizado com as seguintes bibliotecas científicas: NumPy (1.23.5) para operações matriciais, Pandas (1.5.3) para manipulação de dados, e scikit-learn (1.1.1) como base para implementação dos algoritmos de aprendizado de máquina. Complementarmente, foram utilizados LightGBM (4.0.6) e CatBoost (1.0.1) para modelos base em *gradient boosting*, Optuna (2.10.1) para otimização hiperparamétrica, e SHAP (0.39.0) juntamente com LIME (0.2.0.1) para análise de explicabilidade. O gerenciamento de experimentos foi realizado via MLflow (2.11.3), garantindo o rastreamento completo de todos os parâmetros e métricas.

Todos os *frameworks* AutoML avaliados foram executados em suas configurações padrão (*default*), seguindo rigorosamente as recomendações de suas documentações oficiais.

5.4 Critérios de Avaliação da Interpretabilidade e Transparência

Para avaliar especificamente os aspectos de interpretabilidade e transparência dos *frameworks*, foi elaborado um questionário, baseado em perguntas que combinam conceitos de transparência e interpretabilidade propostos por diversos trabalhos da literatura (Arrieta et al., 2020; Brännström, 2023; Carvalho et al., 2019; Love et al., 2023; Mi et al., 2020).

Para criar o questionário de avaliação de *frameworks* AutoML, foram selecionadas cinco categorias para garantir clareza, transparência e interpretabilidade: **Descrição Funcional**, **Análise Estatística**, **Transparência Algorítmica**, **Interpretabilidade** e **Análise Interna**. Essas categorias abordam aspectos que permitem aos usuários compreender e confiar no funcionamento dos *frameworks*, desde a identificação das etapas do *pipeline* até a visualização detalhada da estrutura interna dos modelos, garantindo acesso a informações compreensíveis e detalhadas.

A Descrição Funcional, primeira categoria, avalia se os *frameworks* AutoML cumprem suas funções de forma clara e transparente, identificando as etapas do *pipeline*, compreendendo os componentes de cada etapa, esclarecendo entradas e saídas, e explicando detalhadamente o processo de execução.

A segunda categoria, Análise Estatística, avalia se os *frameworks* AutoML explicam e apresentam claramente os resultados estatísticos e métricas de desempenho. Verifica a apresentação das predições para diferentes classes, o histórico dos dados de treinamento e teste, informações sobre atributos relevantes, a clareza das métricas de desempenho e o uso de métodos visuais, como gráficos e tabelas, para visualização dos resultados.

A Transparência Algorítmica, terceira categoria, avalia a clareza com que os *frameworks* AutoML apresentam informações sobre os algoritmos. Verifica a identificação dos modelos utilizados, o registro e acesso a alertas e erros nos logs, a apresentação clara dos hiperparâmetros e o balanceamento dos dados em termos de classes.

A Interpretabilidade garante que os usuários compreendam os resultados e os processos internos dos modelos gerados pelos *frameworks* AutoML. Esta categoria possui três subcategorias: $pré - modelo$, $no - modelo$ e $pós - modelo$. No $pré - modelo$, é essencial explicar a redução de dimensionalidade e utilizar técnicas específicas do domínio. No $no - modelo$, a ênfase está em modelos com interpretabilidade intrínseca. No $pós - modelo$, é crucial avaliar a diferença entre predições e valores reais e utilizar métodos de interpretabilidade agnósticos ao modelo. Essas práticas garantem confiança do usuário, melhor compreensão dos resultados e identificação de melhorias no processo de modelagem.

A quinta e última categoria, Análise Interna, concentra-se na capacidade dos *frameworks* AutoML de fornecer métodos para visualização da estrutura interna dos modelos. Visualizar a estrutura do modelo auxilia os usuários a entender como os dados são processados e como as decisões são tomadas, como em uma árvore de decisão, onde os ramos e critérios de cada nó são claros. Isso permite melhor interpretação dos resultados, identificação de problemas e áreas de melhoria, promovendo transparência e confiança nos modelos.

A Tabela 5.2 ilustra as perguntas do questionário por categoria, apresentando também o método de avaliação de cada pergunta (não aplicável, parcial ou total), a pontuação de cada resposta e a pontuação máxima (*score*).

Para quantificar as respostas, desenvolvemos um modelo de pontuação S, atribuído para cada categoria analisada, baseado no número de maneiras N (por exemplo, métodos, algoritmos) em que o *framework* satisfaz a questão da categoria.

Tabela 5.2: Modelo de Avaliação de Transparência e Interpretabilidade dos Frameworks de AutoML

Categoria	Pergunta	Pontuação
Descrição Funcional	É possível identificar as etapas do pipeline?	2
	É possível identificar os componentes utilizados em cada etapa?	2
	As entradas e saídas estão claramente definidas?	2
	O funcionamento prático do processo está claro?	2
Análise Estatística	As predições do modelo para as classes (malware e benigno) são apresentadas?	2
	O histórico dos dados de treino e teste é mantido?	2
	O usuário é informado sobre as features mais relevantes?	2
	As métricas de desempenho do modelo são apresentadas?	2
	Há formas de visualizar os resultados (gráficos, tabelas)?	2
Transparência Algorítmica	É possível identificar quais modelos foram utilizados?	2
	É possível acessar avisos, logs e erros do sistema?	2
	Os hiperparâmetros dos modelos gerados são apresentados?	2
	É possível verificar se os dados estão balanceados?	2
Interpretabilidade	A redução de dimensionalidade é justificada? (Pré-modelo)	2
	Há técnicas de redução de dimensionalidade específicas do domínio? (Pré-modelo)	2
	Existem métodos de interpretabilidade global?	2
	O framework oferece modelos com interpretabilidade intrínseca? (Durante o modelo)	2
	É possível avaliar a diferença entre predições e valores reais?	2
	Existem métodos de interpretabilidade independentes do modelo? (Pós-modelo)	2
Análise Interna	Existem métodos para visualizar a estrutura do modelo? (Pós-modelo)	2

A pontuação S é definida como:

$$S = \begin{cases} 0 & \text{se } N = 0 \\ 1 & \text{se } N = 1 \\ 2 & \text{se } N \geq 2 \end{cases}$$

Dessa forma, a pontuação para cada questão varia de 0 a 2 pontos, dependendo do grau de atendimento. Por exemplo, se a resposta à questão for *Não aplicável* ("N"), seu valor será zero (0) pontos. Se a resposta for *parcial* ("P"), seu valor será um (1) ponto. E se a resposta for *total* ("T"), seu valor será dois (2) pontos.

Para gerar uma pontuação máxima (*score*) para um *framework* em determinada categoria, decidimos normalizar a pontuação em uma escala que varia de 0 a 100%.

A fórmula de normalização é a seguinte:

$$S = \left(\frac{\sum_{i=1}^{N} P_i}{N \cdot W} \right) \times 100$$

onde S é a pontuação final normalizada do *framework* para cada categoria, P_i é a pontuação obtida pelo *framework* para a i-ésima questão da categoria, N é o número total de questões nesta categoria e W é a pontuação máxima atribuída por questão.

5.5 Considerações Finais

Este capítulo apresentou a metodologia experimental abrangente desenvolvida para avaliar o framework MH-AutoML em comparação com soluções estabelecidas de AutoML. A abordagem metodológica foi estruturada em quatro pilares principais: seleção criteriosa de conjuntos de dados diversificados, definição transparente de critérios de seleção para os frameworks comparativos, configuração experimental rigorosa e desenvolvimento de um instrumento inovador para avaliação de interpretabilidade e transparência.

A seleção de oito conjuntos de dados heterogêneos, com características variadas em termos de dimensionalidade, balanceamento de classes e complexidade, proporciona uma base sólida para avaliação robusta do desempenho dos frameworks em diferentes cenários. A aplicação consistente de técnicas de amostragem estratificada e balanceamento de classes garante a validade estatística dos resultados obtidos.

Os critérios de seleção adotados para os sete frameworks de AutoML analisados priorizaram representatividade do estado da arte, abrangendo tanto soluções consolidadas quanto abordagens recentes, sempre considerando aspectos de relevância acadêmica,

manutenção ativa e acessibilidade como software livre.

A configuração experimental detalhada, incluindo métricas de avaliação abrangentes (com ênfase especial no Coeficiente de Correlação de Matthews para cenários desbalanceados), parâmetros fixos para garantia de reprodutibilidade, protocolo de validação cruzada estratificada e especificação completa do ambiente computacional, estabelece um padrão rigoroso para experimentação em AutoML que facilita a replicação e validação dos resultados.

Particularmente inovadora é a proposta de um questionário estruturado para avaliação quantitativa de interpretabilidade e transparência, organizado em cinco categorias fundamentais que abrangem desde aspectos funcionais até análise interna dos modelos. Este instrumento representa uma contribuição metodológica significativa para a área, fornecendo uma abordagem sistemática para mensurar características essenciais em aplicações de missão crítica, como é o caso da detecção de *malware*.

A metodologia aqui delineada não apenas serve como base para a avaliação comparativa apresentada nos capítulos subsequentes, mas também estabelece um referencial para futuros estudos comparativos em AutoML, particularmente aqueles focados em domínios especializados que demandam não apenas desempenho preditivo, mas também transparência, interpretabilidade e robustez operacional.

Capítulo 6

Resultados

Este capítulo apresenta uma análise abrangente dos resultados obtidos na avaliação comparativa de oito *frameworks* AutoML para detecção de malware Android. A discussão está organizada em três eixos principais: desempenho preditivo, eficiência computacional e avaliação de transparência/interpretabilidade.

Finalmente, na Seção 6.2, introduzimos uma avaliação inédita de transparência e interpretabilidade baseada em cinco dimensões: descrição funcional, análise estatística, transparência algorítmica, interpretabilidade e análise interna. Nossos resultados revelam que, enquanto todos os frameworks se saíram bem na descrição funcional, apenas MH-AutoML e MLJar atingiram pontuações satisfatórias em interpretabilidade - um aspecto crítico para aplicações de segurança cibernética.

Esta análise multidimensional permite não apenas comparar o desempenho objetivo dos frameworks, mas também avaliar sua adequação para cenários reais de detecção de malware, onde fatores como tempo de resposta e explicabilidade das decisões são tão importantes quanto a acurácia preditiva.

6.1 Desempenho dos Frameworks

Esta seção apresenta uma análise comparativa do desempenho dos *frameworks* AutoML em dois cenários distintos: conjuntos de dados originais (Seção 6.1.1) e conjuntos balanceados com amostras únicas (Seção 6.1.2). A avaliação considera três dimensões fundamentais: (1) *Recall*, que mede a capacidade de detecção de *malware*; (2) o Coeficiente de Correlação de Matthews (MCC), que avalia a qualidade geral da classificação considerando possíveis desbalanceamentos; e (3) os tempos de execução, que permitem analisar o *trade-off* entre desempenho preditivo e eficiência computacional.

Para facilitar a interpretação dos resultados, empregamos mapas de calor que representam visualmente o desempenho relativo dos oito *frameworks* em cada conjunto de dados. Esta abordagem oferece várias vantagens analíticas:

- Permite identificar rapidamente padrões de desempenho através de gradientes de cores intuitivos
- Facilita a comparação entre múltiplos *frameworks* e conjuntos de dados simultaneamente
- Destaca casos de desempenho excepcional ou deficiente

6.1.1 Conjuntos de Dados Originais

A Figura 6.1 apresenta os resultados das análises dos *frameworks* de AutoML em relação à métrica de *Recall* entre os diferentes conjuntos de dados analisados.

Figura 6.1: Mapa de calor do *Recall* dos 8 *frameworks* AutoML.

No conjunto de dados **KronoDroid Emulador**, os *frameworks* apresentaram resultados bastante próximos, com o LightAutoML alcançando o maior *Recall* (96,97%),

seguido por AutoGluon (96,93%) e MH-AutoML (96,73%). O desempenho menos expressivo foi obtido pelo HyperGBM (94,03%). Padrão similar foi verificado no conjunto **KronoDroid Real**, com o AutoGluon atingindo o melhor resultado (97,51%), acompanhado por Auto-Sklearn (97,29%) e AutoPyTorch (97,10%). Neste caso, o menor *Recall* foi registrado pelo MH-AutoML (96,59%).

No conjunto de dados **Androcrawl**, o MH-AutoML superou os demais *frameworks* com *Recall* de 95,41%, seguido por Auto-Sklearn (93,52%) e LightAutoML (92,44%). O AutoGluon apresentou o desempenho mais modesto (91%), sugerindo possíveis limitações em conjuntos de dados com características específicas de *malware*. Uma situação particular ocorreu no conjunto **Android Permissions**, no qual os *frameworks* AutoPyTorch (99,94%) e Auto-Sklearn (99,19%) alcançaram os melhores resultados, enquanto o MH-AutoML obteve o pior desempenho (53,79%). Essa discrepância decorre do método de cálculo do *Recall* como média entre classes, sendo que o conjunto contém apenas amostras de *malware*. Na realidade, o MH-AutoML alcançou 96% de *Recall* para a classe de *malware*, desempenho equivalente aos melhores *frameworks*, enquanto a métrica agregada foi distorcida pela ausência de dados benignos.

Nos conjuntos da família Defensedroid, os resultados mostraram padrões distintos. No conjunto **Defensedroid PRS**, o MH-AutoML obteve o melhor *Recall* (91,73%), enquanto o Auto-Sklearn registrou o menor valor (88,43%). Já no conjunto **Defensedroid API Calls**, o MH-AutoML novamente se destacou (94,02%), com o TPOT apresentando o desempenho mais baixo (92,59%). O conjunto **Drebin** caracterizou-se por elevados índices de *Recall* em todos os *frameworks*, com destaque para o MH-AutoML (98,51%), seguido por MLJar (97,95%) e LightAutoML (97,86%). Mesmo o pior resultado, obtido por HyperGBM (97,06%), manteve-se em patamar competitivo.

O conjunto **Adroit** revelou-se particularmente desafiador, com o MH-AutoML alcançando o melhor desempenho (86,93%) e o HyperGBM o mais baixo (76,76%). Os demais *frameworks* apresentaram resultados próximos de 79%, indicando maior dificuldade das técnicas AutoML para esse conjunto específico.

Padrão similar foi observado no conjunto **MH-100-4k Features**, no qual o MH-AutoML novamente liderou (94,22%), acompanhado por MLJar (92,86%) e Auto-Sklearn (90,85%). Os menores valores foram registrados por TPOT (86,67%) e HyperGBM (86,52%).

De modo geral, a análise demonstra que o MH-AutoML se mostrou o *framework* mais consistente para maximizar o *Recall*, liderando em seis dos nove conjuntos de dados analisados. Os *frameworks* AutoPyTorch e Auto-Sklearn apresentaram desempenho excepcional em cenários específicos, como no **Android Permissions**, mas com

menor consistência em outros conjuntos. O HyperGBM registrou os piores resultados na maioria dos casos, exceto no **KronoDroid Real** (96,34%), no qual seu desempenho foi satisfatório. Já o LightAutoML e o AutoGluon mantiveram desempenho competitivo na maioria dos conjuntos, especialmente em **Drebin** e **KronoDroid**, demonstrando boa robustez nas diferentes condições testadas.

A análise do Coeficiente de Correlação de Matthews (MCC), ilustrada na Figura 6.2, revelou padrões distintos de desempenho entre os *frameworks* de AutoML nos diferentes conjuntos de dados avaliados.

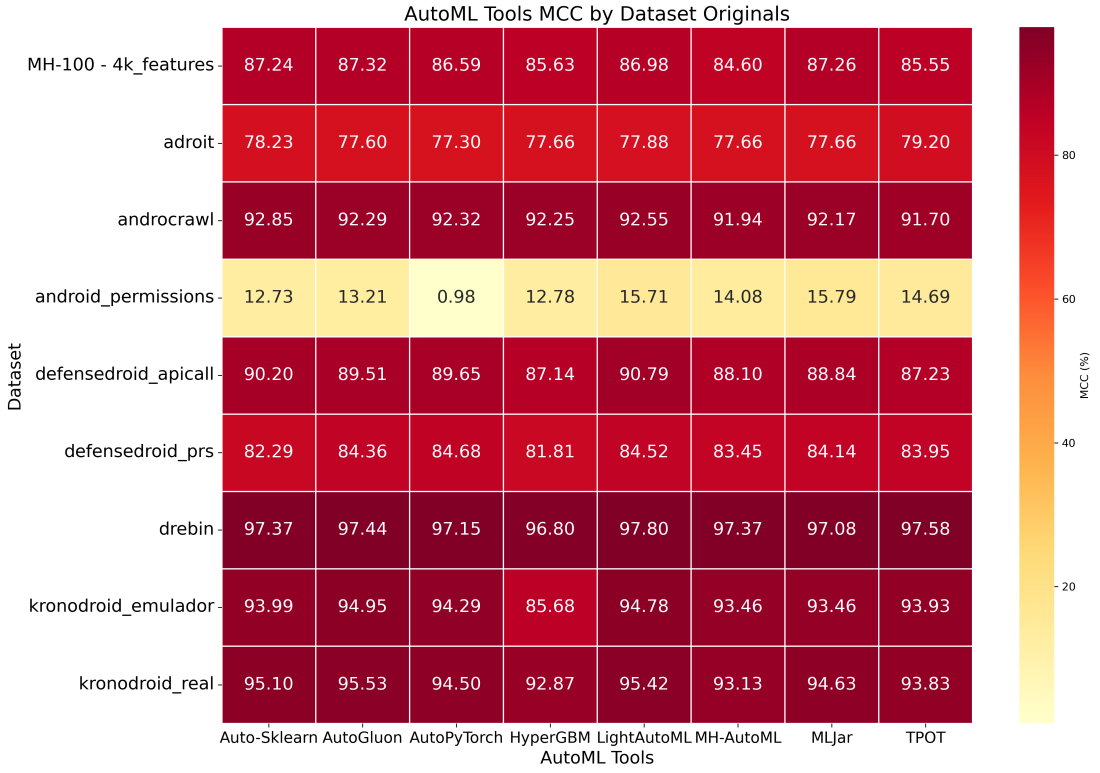


Figura 6.2: Mapa de calor do MCC dos 8 frameworks AutoML.

No conjunto de dados **KronoDroid Emulador**, os valores de MCC apresentaram variação relativamente pequena entre os *frameworks*, com destaque para o LightAutoML (94,78%), seguido por AutoGluon (94,95%) e AutoPyTorch (94,29%). Todos os *frameworks* seguiram essa valoração de MCC, com excessão ao HyperGBM, que apresentou desempenho de 85,68%. Um padrão semelhante foi observado no conjunto **KronoDroid Real**, no qual AutoGluon (95,53%), LightAutoML (95,42%) e Auto-Sklearn (95,10%) obtiveram os melhores resultados, enquanto o MH-AutoML registrou

o valor mais baixo (93,13%).

No conjunto **Androcrawl**, os *frameworks* apresentaram desempenhos bastante equilibrados, com variação máxima de apenas 1,15% entre o melhor desempenho (Auto-Sklearn, com 92,85%) e o pior (TPOT, com 91,70%). Uma situação radicalmente distinta foi observada no conjunto **Android Permissions**, onde todos os *frameworks* obtiveram desempenho excepcionalmente baixo, com valores de MCC variando entre 0,98% (AutoPyTorch) e 15,79% (MLJar). Esse comportamento atípico sugere que as características específicas deste conjunto de dados representam um desafio particular para a métrica MCC, possivelmente devido à distribuição peculiar das classes ou à natureza dos atributos.

Nos conjuntos da família Defensedroid, os resultados demonstraram maior consistência. No conjunto **Defensedroid PRS**, o AutoPyTorch obteve o melhor MCC (84,68%), seguido por LightAutoML (84,52%) e AutoGluon (84,36%). O HyperGBM apresentou o desempenho mais fraco (81,81%). Já no conjunto **Defensedroid API Calls**, o LightAutoML destacou-se com 90,79%, acompanhado por Auto-Sklearn (90,20%) e AutoPyTorch (89,65%). O TPOT registrou o menor valor (87,23%), mas ainda se mantendo em um patamar razoável.

O conjunto **Drebin** caracterizou-se por elevados valores de MCC em todos os *frameworks*, com destaque para LightAutoML (97,80%), TPOT (97,58%) e AutoGluon (97,44%). Mesmo o pior resultado, obtido por HyperGBM (96,80%), manteve-se em nível excelente. No conjunto **Adroit**, os *frameworks* apresentaram desempenhos mais modestos e bastante similares, com o TPOT alcançando o melhor MCC (79,20%) e o AutoPyTorch o valor mais baixo (77,30%).

No conjunto **MH-100-4k Features**, o AutoGluon obteve o melhor desempenho (87,32%), seguido por MLJar (87,26%) e Auto-Sklearn (87,24%). O MH-AutoML registrou o valor mais baixo (84,60%), situando-se significativamente abaixo de seus resultados obtidos em outros conjuntos.

De modo geral, a análise do MCC revelou que o LightAutoML e o AutoGluon foram os *frameworks* mais consistentes, apresentando bons resultados na maioria dos conjuntos de dados. O HyperGBM tendeu a apresentar os piores desempenhos, enquanto o MH-AutoML mostrou variações entre os conjuntos, revelando um desempenho intermediário. A exceção notável foi o conjunto **Android Permissions**, no qual todos os *frameworks* apresentaram desempenho abaixo do esperado, sugerindo que esse conjunto possui características particulares que desafiam a eficácia da métrica MCC.

Por fim, a avaliação dos tempos de execução, apresentada na Figura6.3, revelou disparidades significativas entre os diferentes *frameworks* de AutoML, com variações

de até três ordens de grandeza em alguns conjuntos de dados.

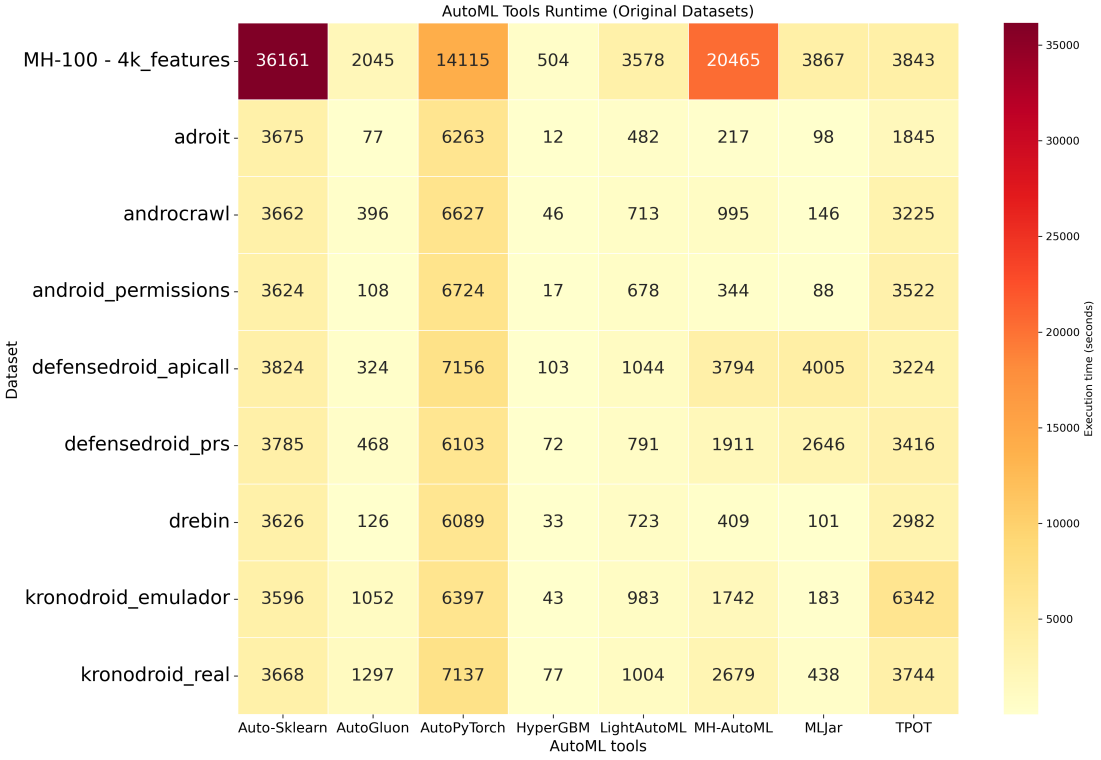


Figura 6.3: Dados de tempo de execução dos frameworks AutoML.

O HyperGBM destacou-se como o mais rápido em todos os cenários, com tempos que variaram de meros 12 segundos **Adroit** a 8 minutos e 24 segundos **MH-100-4k Features**. Em posição intermediária, LightAutoML, AutoGluon e MLJar apresentaram tempos competitivos, geralmente abaixo de 30 minutos na maioria dos *datasets*.

Os *frameworks* baseados em redes neurais AutoPyTorch e otimização abrangente Auto-Sklearn, TPOT exibiram os maiores tempos de execução, frequentemente ultrapassando 1 hora e 30 minutos. Caso extremo ocorreu no *dataset* **MH-100-4k Features**, onde Auto-Sklearn demandou 10 horas e 2 minutos, seguido por MH-AutoML com 5 horas e 41 minutos. Este comportamento sugere que conjuntos de dados complexos e de alta dimensionalidade impõem desafios computacionais particulares a estas abordagens.

O MH-AutoML apresentou desempenho heterogêneo. Enquanto em conjuntos menores, como **Adroit** (3 minutos 37 segundos) e **Android Permissions** (5 minutos 44 segundos), manteve tempos razoáveis, no **Defensedroid API Calls** (1 hora 3 minutos) e especialmente no **MH-100-4k Features** (5 horas 41 minutos) mostrou limitações de escalabilidade. Esse comportamento pode ser atribuído diretamente à sua arquitetura

sofisticada de avaliação, diferentemente de *frameworks* com *pipelines* mais simples como HyperGBM e LightAutoML.

No geral, HyperGBM, LightAutoML, AutoGluon, MLJar foram os *frameworks* mais rápidos, com execuções deaté 30 minutos. MH-AutoML (na maioria dos casos) e TPOT tiveram tempos intermediários (30-90 minutos) e Auto-Sklearn e AutoPyTorch apresentaram tempos acima de 90 minutos. A análise comparativa dos tempos de execução demonstra claramente que *frameworks* com arquiteturas mais simples e pipelines lineares, como HyperGBM e LightAutoML, tendem a apresentar tempos de processamento significativamente menores. Esta vantagem operacional é particularmente evidente em conjuntos de dados de grande dimensionalidade, onde a simplicidade algorítmica se traduz em eficiência computacional.

Contudo, é notável que o MH-AutoML, apesar de sua arquitetura complexa que incorpora métodos sofisticados de interpretabilidade e otimização hierárquica de hiperparâmetros, mantém-se competitivo em muitos cenários, especialmente em conjuntos de dados de dimensionalidade moderada. O MH-AutoML compensa seu maior custo computacional com análises mais profundas e resultados potencialmente mais robustos, particularmente em problemas complexos onde a mera velocidade de execução não é o principal critério. Esta dualidade revela um equilíbrio fundamental no ecossistema AutoML: enquanto soluções simplificadas são ideais para aplicações que demandam rapidez, abordagens mais elaboradas como o MH-AutoML oferecem valor analítico adicional que pode ser crucial em cenários onde a compreensão do modelo e a fineza de ajuste são prioritárias.

6.1.2 Conjuntos de Dados Balanceados com Amostras Únicas

Nesta seção examinamos como o balanceamento de dados afetou o desempenho. As Figuras 6.4, 6.5 e 6.6 apresentam mapas de calor com os resultados de desempenho dos 8 frameworks AutoML, destacando as métricas de *Recall*, *MCC* (Coeficiente de Correlação de Matthews) e tempo de execução, respectivamente, para os 9 conjuntos de dados balanceados com amostras únicas.

Os resultados apresentados nas Figuras 6.4 e 6.5 mostram variações significativas no desempenho dos *frameworks* de AutoML nos conjuntos testados, tanto nas métricas de avaliação (*Recall* e MCC) quanto no tempo de execução. Alguns *frameworks*, como LightAutoML, demonstraram desempenho consistente e robusto, frequentemente alcançando as melhores ou próximas das melhores métricas em diversos conjuntos, aliado a tempos de execução competitivos. Em contraste, *frameworks* como TPOT e AutoPy-

Figura 6.4: *Recall* dos *frameworks* AutoML por Conjunto de Amostras Únicas.

Torch exibiram longos tempos de execução, mas nem sempre converteram essa maior complexidade em melhores resultados.

No conjunto **Kronodroid Emulator**, LightAutoML destacou-se pelo equilíbrio geral, alcançando *Recall* de 91,49% e MCC de 86,09%. No conjunto **Kronodroid Real**, o mesmo *framework* também alcançou um dos melhores resultados, com *Recall* de 93,16%, próximo ao do MLJar (93,74%), e MCC de 89,66%. LightAutoML foi, no geral, o *framework* mais eficiente neste cenário, enquanto AutoGluon e MLJar apresentaram resultados similares com tempos menores.

No conjunto **Androcrawl**, observamos métricas de *Recall* muito elevadas para todos os *frameworks*, com todos os valores entre 95% e 97%. AutoGluon alcançou o maior valor (96,56%) enquanto o HyperGBM mostrou o melhor MCC (87,20%). LightAutoML e AutoGluon também mantiveram resultados consistentes.

No conjunto **Android Permissions**, nenhum *framework* apresentou métricas particularmente altas. A exceção foi o Auto-Sklearn, que alcançou o melhor *Recall* (91,41%), mas seu MCC foi de apenas 1,09%, indicando que o modelo treinado previa quase todas as amostras como *malware*, resultando em grande número de falsos positivos e

Figura 6.5: MCC dos *frameworks* AutoML por Conjunto de Amostras Únicas.

levando a um MCC baixo apesar do *Recall* alto. Neste conjunto, também vemos um desempenho anômalo do TPOT, com MCC de -8,80%, indicando que o modelo estava invertendo os rótulos, causando desempenho muito pobre (negativo), o que também é evidenciado por seu baixo *Recall*. O baixo desempenho geral pode ser atribuído à complexidade intrínseca do conjunto, dificultando o ajuste dos modelos, com classificações quase aleatórias.

Para o conjunto **DefenseDroid PRS**, os *frameworks* MLJar e AutoGluon alcançaram os melhores resultados para *Recall* e MCC, com MLJar obtendo *Recall* de 93,17% e MCC de 85,47%, enquanto AutoGluon alcançou 93,08% e 85,55%, respectivamente. Para o conjunto **DefenseDroid API Calls**, MH-AutoML alcançou o maior *Recall* com 89,75%, seguido por Auto-Sklearn e LightAutoML, que alcançaram *Recall* de 89,63%. Para MCC, LightAutoML alcançou o maior valor com 85,26%. Apesar do alto *Recall*, MH-AutoML obteve MCC baixo de 79,74%.

No conjunto **Drebin**, MH-AutoML liderou em *Recall* (93,10%) com tempo de execução de 4 minutos e 21 segundos. LightAutoML registrou o melhor MCC (89,18%) com tempo de execução de 9 minutos e 22 segundos.

O conjunto **Adroit** apresentou desafios significativos. MH-AutoML liderou em *Recall* (53,19%), mas o desempenho geral dos *frameworks* foi baixo, sugerindo alta complexidade ou características não representativas no conjunto. Nenhum *Framework* alcançou desempenho significativo.

Para o conjunto **MH-100 4k features**, o *framework* AutoGluon alcançou o maior *Recall* (96,68%), indicando alta capacidade de identificar corretamente amostras positivas. Porém, seu MCC (76,52%) sugere que a classificação geral pode não estar bem balanceada. MH-AutoML também mostrou alto *Recall* (95,66%) e MCC maior (78,36%), sugerindo melhor correlação entre previsões e rótulos reais. Auto-Sklearn alcançou *Recall* de 95,66%, similar ao MH-AutoML, mas com MCC ligeiramente maior (80,95%). TPOT alcançou *Recall* de 96,43%, mas com MCC de 74,30%, indicando que sua classificação pode estar favorecendo excessivamente uma das classes. AutoPyTorch teve *Recall* de 96,02% e MCC de 78,81%, mantendo bom equilíbrio entre essas métricas. LightAutoML alcançou *Recall* de 96,38% e MCC de 80,07%, sendo um dos melhores equilíbrios entre *Recall* e correlação de previsões. HyperGBM, apesar de mostrar *Recall* de 95,66%, tem MCC ligeiramente menor (77,86%). MLJar alcançou *Recall* de 96,58% e MCC de 76,88%.

Na Figura 6.6 observamos o tempo de execução dos *frameworks*.

AutoGluon e HyperGBM mantiveram tempos de execução reduzidos em todos os conjuntos. A velocidade excepcional do *framework* HyperGBM é resultado do treinamento limitado a quatro modelos (*XGBoost*, *CatBoost*, *LightGBM* e *HistGradientBoosting*). Por outro lado, os *frameworks* Auto-Sklearn, AutoPyTorch e TPOT apresentaram tempos de execução significativamente maiores sem alcançar desempenho destacado nas métricas. Isto pode ser explicado por suas estratégias de otimização altamente complexas: meta-aprendizado com otimização bayesiana para Auto-Sklearn, otimização bayesiana para AutoPyTorch e programação genética para TPOT.

O *framework* MH-AutoML mostrou-se competitiva em alguns cenários, mantendo bons tempos de execução. Porém, seu desempenho pode ser impactado em conjuntos com maior número de características devido ao tempo adicional requerido por suas técnicas de seleção de características. Adicionalmente, a ausência de um ensemble final com os modelos treinados pode explicar métricas mais baixas comparado a *frameworks* que utilizam esta abordagem.

No geral, LightAutoML e AutoGluon provaram ser os *frameworks* mais equilibrados, combinando altos valores de métricas com baixos tempos de execução. LightAutoML teve os melhores resultados para MCC, mantendo tempos de execução baixos.

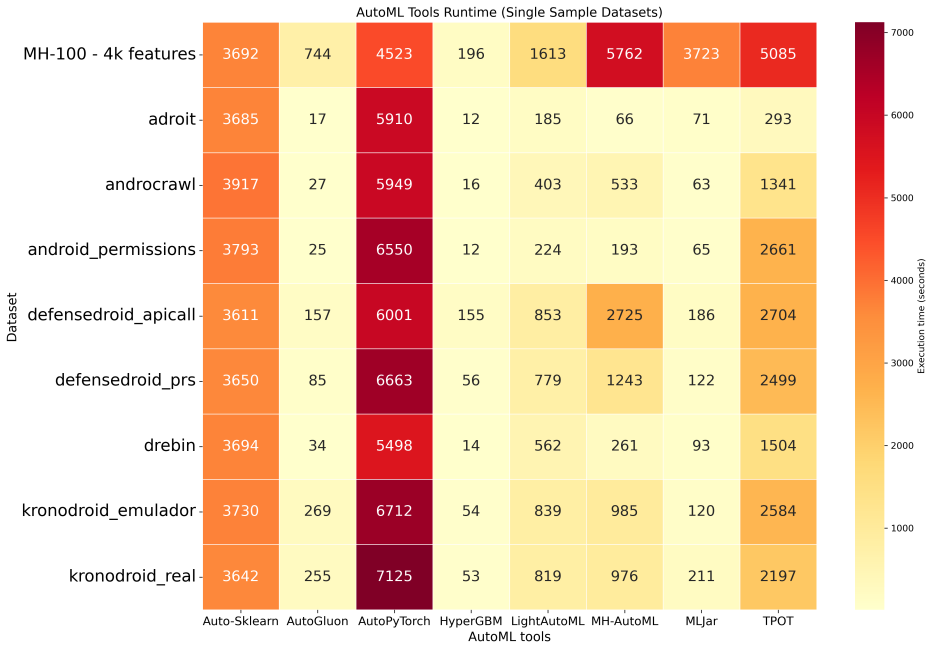


Figura 6.6: Tempo de Execução dos frameworks AutoML.

AutoGluon, Auto-Sklearn e MH-AutoML também mostraram bom desempenho, com destaque para o baixo tempo de AutoGluon e MH-AutoML, embora o tempo de Auto-Sklearn permaneça alto. Vale destacar o alto desempenho de MH-AutoML na métrica *Recall*, mas seu desempenho em MCC é mediano, sugerindo que este *framework* constrói modelos que têm maior facilidade em classificar malware.

O *framework* LightAutoML destacou-se por seu excelente desempenho geral, resultado de seu pipeline eficiente que combina diferentes modelos base (CatBoost, LightGBM, XGBoost, Random Forest e Redes Neurais) através da técnica de *blending*. O equilíbrio entre tempo e desempenho é alcançado graças à sua estratégia de *early stopping* e otimização paralela, garantindo resultados competitivos com tempos de execução reduzidos.

O *framework* AutoGluon também apresentou desempenho altamente competitivo, atribuído à sua estratégia de *stacking*. O *framework* utiliza ensembles ponderados que combinam inteligentemente múltiplos modelos base. Adicionalmente, frequentemente é um dos *frameworks* mais rápidas devido à adoção de hiperparâmetros fixos para todos os modelos, eliminando a necessidade de otimizações demoradas e garantindo eficiência no tempo de execução.

6.1.3 Comparação entre Conjuntos Originais e Amostras Únicas Balanceadas

A análise dos resultados para os conjuntos balanceados com amostras únicas revela diferenças significativas no desempenho dos *frameworks* AutoML quando comparados aos conjuntos originais. Em geral, observa-se que o balanceamento dos dados tende a melhorar o desempenho em termos de *Recall* e MCC para a maioria dos *frameworks*, embora com variações dependendo do conjunto e do *framework* utilizado.

Para o conjunto **KronoDroid Emulator**, por exemplo, o *Recall* dos *frameworks* aumentou consideravelmente após o balanceamento, com destaque para HyperGBM, que alcançou *Recall* de 91,69%, comparado a 94,03% no conjunto original. Porém, o *MCC* apresentou ligeira redução, indicando que embora o balanceamento tenha melhorado a capacidade de detecção das classes minoritárias, a precisão geral pode ter sido comprometida. LightAutoML manteve desempenho consistente, com *Recall* de 91,49% e MCC de 86,09%, demonstrando robustez mesmo em cenários balanceados.

No conjunto **Androcrawl**, o balanceamento resultou em aumento significativo de *Recall* para todos os *frameworks*, com AutoGluon atingindo 96,56%, comparado a 91,00% no conjunto original. MH-AutoML também apresentou desempenho notável, com *Recall* de 96,13%. Porém, o MCC permaneceu relativamente estável, sugerindo que o balanceamento não afetou significativamente a correlação entre previsões e valores reais.

Um caso interessante é o conjunto **Android Permissions**, onde o balanceamento resultou em melhoria drástica no *Recall* para alguns *frameworks*, como Auto-Sklearn, que alcançou 91,41%, comparado a 99,19% no conjunto original. ' Porém, o MCC foi extremamente baixo (0,0109), indicando que o balanceamento pode ter introduzido ruído ou reduzido a capacidade de generalização do modelo. TPOT, por outro lado, apresentou desempenho muito inferior após o balanceamento, com *Recall* de apenas 6,79% e *MCC* negativo, sugerindo que este *framework* pode não ser adequado para conjuntos balanceados com amostras únicas.

Para o conjunto **Drebin**, o balanceamento resultou em aumento do *Recall* para MH-AutoML, que atingiu 93,10%, comparado a 98,51% no conjunto original. Porém, o MCC diminuiu ligeiramente, indicando possível perda de precisão. LightAutoML manteve desempenho sólido, com *MCC* de 89,18%, o maior entre todos os *frameworks*.

Quanto ao tempo de execução, o balanceamento dos dados tendeu a reduzir o tempo necessário para treinamento dos *frameworks*. Por exemplo, no conjunto **KronoDroid Emulator**, o tempo de execução de AutoGluon caiu de 17 minutos para 4 minutos e 29

segundos. Porém, *frameworks* como Auto-Sklearn e AutoPyTorch continuaram a exigir tempos de execução prolongados, mesmo após o balanceamento.

Em resumo, o balanceamento de amostras únicas tende a melhorar o *Recall*, especialmente para conjuntos com classes desbalanceadas, mas pode comprometer o MCC em alguns casos. *Frameworks* como LightAutoML e AutoGluon demonstraram consistência e robustez tanto nos conjuntos originais quanto nos balanceados, enquanto outras, como TPOT, apresentaram desempenho inferior após o balanceamento. A escolha do *framework* de AutoML deve, portanto, considerar não apenas as métricas de desempenho, mas também o impacto do balanceamento nos resultados e no tempo de execução.

6.2 Avaliação de Transparência e Interpretabilidade

Na avaliação dos *frameworks* de AutoML com base no questionário proposto, observou-se que, na categoria de *Descrição Funcional*, todos os *frameworks* alcançaram a pontuação máxima de 100 pontos, indicando clareza e transparência em suas funcionalidades. Na categoria de *Análise Estatística*, o MH-AutoML e o MLJar lideraram com 100 pontos, seguidos pelo AutoGluon e HyperGBM, ambos com 80 pontos. LightAutoML, Auto-Sklearn, TPOT e AutoPyTorch obtiveram pontuações entre 70 e 60 pontos. Com exceção do MH-AutoML e do MLJar, todos os *frameworks* ou não mantêm um histórico dos dados utilizados para treinar e testar o modelo, ou não o apresentam, ou apresentam apenas parcialmente as características mais relevantes ao usuário.

Em termos de *Transparência Algorítmica*, apenas o MH-AutoML alcançou 100 pontos, seguido por AutoGluon, AutoPyTorch, LightAutoML, HyperGBM e MLJar, cada um com 75 pontos. Auto-Sklearn e TPOT apresentaram espaço para melhorias, com apenas 50 pontos cada. As principais deficiências desses *frameworks* estão relacionadas à falta de transparência quanto ao balanceamento de classes e à ausência de registros (*logs*) do sistema.

Na categoria de *Interpretabilidade*, o MH-AutoML e o MLJar foram os melhores avaliados, com 58,33 pontos, destacando-se por fornecer modelos e resultados de fácil compreensão. O HyperGBM também obteve uma boa pontuação de 50 pontos. LightAutoML e AutoGluon receberam 41,6 pontos cada. AutoPyTorch apresentou pontuação mediana de 33,33 pontos, enquanto Auto-Sklearn obteve 25 pontos. TPOT teve a menor pontuação, com 8,33 pontos, indicando desafios significativos de interpretabilidade. As principais deficiências desses *frameworks* incluem a ausência de opções para utilizar exclusivamente métodos de interpretabilidade intrínsecos, métodos específicos de domínio,

explicações para a escolha dos métodos de redução de dimensionalidade e ausência de métodos de interpretabilidade independentes do modelo, como SHAP e LIME.

Por fim, na categoria de *Análise Interna*, TPOT, HyperGBM e MLJar se destacaram com 100 pontos cada, demonstrando capacidade de apresentar em detalhes a estrutura dos modelos utilizados, como árvores de decisão, por exemplo. AutoGluon, MH-AutoML e Auto-Sklearn receberam 50 pontos nesta categoria, indicando uma apresentação moderada da estrutura dos modelos. Por outro lado, AutoPyTorch e LightAutoML apresentaram grande margem para melhorias, com pontuação 0, refletindo a ausência de detalhamento na apresentação da estrutura dos modelos utilizados.

A Figura 6.7 ilustra os resultados da avaliação dos *frameworks* de AutoML em relação às 5 categorias focadas em interpretabilidade e transparência.

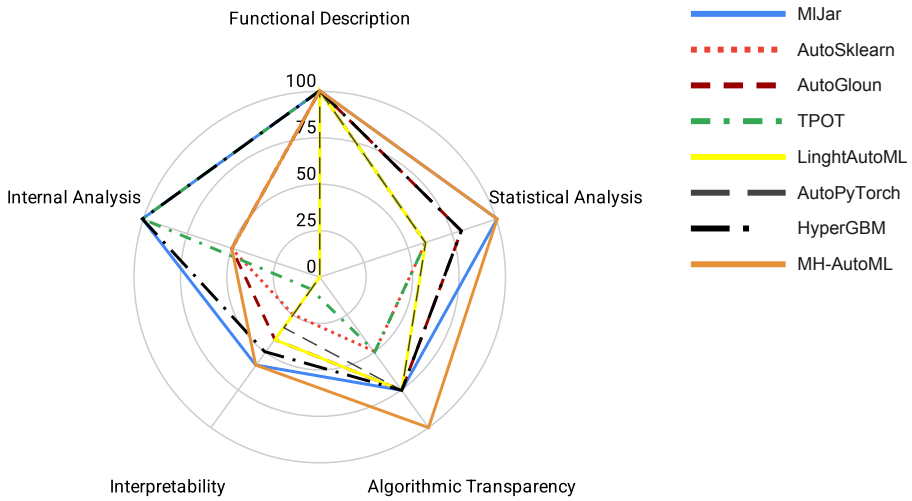


Figura 6.7: Avaliação dos *frameworks* de AutoML

Observa-se na Figura 6.7 que alguns *frameworks* se destacam na descrição funcional, mas sua capacidade de fornecer modelos compreensíveis e de visualizar a estrutura interna ainda requer melhorias substanciais. Adicionalmente, também necessitam de aprimoramentos para aumentar a confiança nos resultados e facilitar o entendimento das decisões tomadas pelos modelos (interpretabilidade), especialmente em aplicações críticas como a detecção de malware em dispositivos Android.

Capítulo 7

Conclusão

O *framework* MH-AutoML se destaca significativamente no cenário atual de AutoML por conseguir conciliar de maneira equilibrada dois objetivos tradicionalmente conflitantes: alto desempenho preditivo e robustos recursos de interpretabilidade/transparência. Esse equilíbrio, alcançado sem comprometer significativamente a eficiência computacional, posiciona o *framework* como uma solução alinhada com os princípios de XAI (*Explainable Artificial Intelligence*) propostos pela DARPA.

A MH-AutoML oferece recursos avançados de interpretabilidade, transparência e depuração em comparação com outros *frameworks* AutoML, incluindo métodos de seleção de características e técnicas específicas por modelo para classificação de *malware* Android. Sua integração com MLflow permite versionamento e rastreamento completo do pipeline AutoML, proporcionando uma análise detalhada do processo de modelagem. Na comparação com sete *frameworks* AutoML líderes (AutoGluon, Auto-Sklearn, TPOT, AutoPyTorch, MLJAR, HyperGBM e LightAutoML) no domínio de detecção de malware Android, a MH-AutoML demonstra potencial competitivo, combinando:

1. **Transparência e Usabilidade**: Juntamente com o MLJAR, destaca-se por sua interface amigável para não especialistas (Figura 6.7), com o diferencial do rastreamento completo via MLflow.

2. **Eficiência Computacional**: Embora HyperGBM e AutoGluon sejam os *frameworks* mais rápidos - a primeira utilizando técnicas como *early stopping* e a segunda empregando hiperparâmetros fixos - a MH-AutoML mantém tempos de execução competitivos.

3. **Desempenho Preditiv**: Enquanto AutoGluon e LightAutoML lideram em métricas de classificação, a MH-AutoML oferece resultados consistentes em todos os

conjuntos de dados testados.

A MH-AutoML supera os desafio de implementar: (i) métodos explicáveis intrínsecos (como *Decision Trees* e *Random Forests*), técnicas de pós-processamento para extrair explicações de modelos complexos, e monitoramento integrado do processo de modelagem. Esta capacidade de conciliar explicabilidade com desempenho preditivo coloca a MH-AutoML na vanguarda das iniciativas de XAI, particularmente na detecção de *malware* Android, onde tanto a precisão quanto a transparência são essenciais.

7.1 Trabalhos Futuros

A MH-AutoML apresenta várias oportunidades de aprimoramento e expansão de funcionalidades. Os principais eixos de desenvolvimento futuro incluem:

- Integração com *Large Language Models* (LLMs) para geração automática de relatórios explicativos em linguagem natural, tradução de métricas técnicas em *insights* acionáveis e sistema de Q&A interativo sobre os resultados.

- Otimização da seleção de características, através da incorporação do *framework* **MH-FSF**[1] para avaliação sistemática de métodos de seleção de características, *benchmarking* automático de técnicas (filtros, *wrappers*, *embedded*) e seleção adaptativa baseada no tipo de dado e modelo.

- Paralelização e Escalabilidade, aplicando GPU *Acceleration*, para o treino de modelos *Deep Learning* e matrizes grandes; *Dask Distributed*, para o pré-processamento em *datasets* massivos; *Ray Tune*, para otimização hiperparâmetros distribuída; e MPI Interface, para comunicação entre nós HPC.

- Interface Visual Interativa, através do desenvolvimento de interface web com editor *drag-and-drop* de pipelines, templates pré-configurados para domínios específicos e Visualização 3D de espaços de características.

- Suporte a Multi-formatos, para além de dados tabulares, como séries temporais (integração com Prophet e Kats), textos (pipelines de NLP com spaCy e HF Transformers), imagens para visão computacional e Grafos (processamento com DGL e PyG)

[1]

- Integração com Plataformas Externas, como armazenamento em cloud (S3, GCS, Azure Blob), *frameworks* de BI (Tableau, PowerBI) e bibliotecas de segurança adversarial (ART - *Adversarial Robustness Toolbox*)
- Integração com segurança ML, com a avaliação da robustez adversarial com ART (suporte para scikit-learn, LightGBM, CatBoost), geração de amostras adversariais (FGSM, PGD, DeepFool), monitoramento de *robust accuracy* como métrica adicional nos experimentos e defesa contra perturbações com técnicas de purificação e detecção

7.2 Desafios Técnicos Encontrados

Durante o desenvolvimento e validação da MH-AutoML, diversos desafios técnicos e conceituais foram enfrentados, exigindo soluções específicas para garantir um *pipeline* robusto, explicável e reprodutível. As dificuldades incluíram incompatibilidades entre bibliotecas, limitações na interpretabilidade de modelos complexos, problemas de desempenho com grandes volumes de dados, desafios na rastreabilidade de experimentos e mudanças na API da biblioteca SHAP. Esta seção apresenta os principais obstáculos encontrados, juntamente com as soluções adotadas.

Incompatibilidades e Modelos Não Suportados

Alguns algoritmos apresentaram restrições à aplicação de técnicas de explicabilidade como SHAP e LIME. Modelos de votação majoritária, por exemplo, não permitiam a computação direta de valores de Shapley. A solução consistiu em substituir a votação majoritária por votação ponderada, sempre que possível, ou realizar a explicação separadamente para cada modelo base.

No caso de técnicas de redução de dimensionalidade como PCA, observou-se a perda de correspondência direta entre atributos originais e componentes transformados, dificultando a explicação. Para resolver esse problema, foi utilizado um modelo probabilístico de PCA, que permitiu estimar a contribuição de cada componente considerando a distribuição condicional das variáveis.

Desafios com a API do SHAP

A biblioteca SHAP passou por mudanças significativas a partir da versão 0.20, afetando diretamente a forma de geração de gráficos explicativos. A nova API exigia uma estrutura diferente de entrada e modificações na ordem dos parâmetros. Além disso, os

modelos retornavam os valores de explicação em diferentes formatos — arrays tridimensionais ou listas de arrays —, variando conforme o algoritmo utilizado. A solução foi implementar uma lógica de detecção automática do tipo de estrutura e selecionar a abordagem adequada para cada modelo.

Exibição dos Nomes das Características em Gráficos

Inicialmente, os gráficos gerados mostravam nomes genéricos como "Feature 0", o que prejudicava a interpretação dos resultados. Isso ocorria devido à forma como os nomes dos atributos eram passados ou inferidos pela biblioteca. A correção envolveu o uso explícito dos nomes das variáveis no momento da criação dos objetos explicativos e nos métodos de visualização, além de mecanismos de *fallback* para cenários em que a API falhava em reconhecer corretamente os atributos.

Gerenciamento de Dependências

A integração de diversas bibliotecas modernas gerou conflitos de versão, especialmente em ambientes que utilizavam diferentes versões do Python entre 3.5 e 3.9. Para garantir a compatibilidade e estabilidade, optou-se pela padronização do ambiente para Python 3.8.10 e pela definição de um arquivo `requirements.txt` cuidadosamente curado, assegurando o funcionamento uniforme do sistema em diferentes plataformas.

Desempenho Computacional e Interpretabilidade

O cálculo exato dos valores de Shapley é computacionalmente intensivo. Para contornar essa limitação, utilizaram-se aproximações baseadas em amostragem via Monte Carlo. Além disso, foram aplicadas estratégias de otimização, como o uso de bibliotecas para paralelização de tarefas e particionamento de dados, além da substituição de estruturas de dados pesadas por alternativas mais leves.

No que diz respeito à rastreabilidade, observou-se que soluções automatizadas geravam logs pouco informativos. Foi, portanto, necessário adotar uma abordagem manual e modular, com uso de convenções de nomenclatura e organização hierárquica de experimentos para facilitar o monitoramento e reuso de resultados.

Compatibilidade entre Modelos de ML

Um dos desafios mais significativos foi garantir que todos os modelos suportados pelo MH-AutoML funcionassem corretamente com os métodos de interpretabilidade SHAP e

LIME. Foram realizados testes extensivos com os cinco algoritmos principais do sistema:

- **RandomForestClassifier**: Apresentou desafios específicos com arrays numpy 3D de shape $(1, n\_features, n\_classes)$, exigindo seleção da classe correta via `shap_values[..., 1]` para classificação binária.
- **DecisionTreeClassifier**: Similar ao RandomForest, mas com estrutura mais simples. Requer tratamento especial para garantir que os *feature names* sejam preservados corretamente nos gráficos SHAP.
- **ExtraTreesClassifier**: Compatível com *TreeExplainer* do SHAP, mas com as mesmas limitações de formato de dados dos modelos baseados em árvores do scikit-learn.
- **LGBMClassifier**: Retorna lista de arrays para cada classe, diferente dos modelos scikit-learn. Requer seleção explícita da classe via `shap_values[1]` para a classe positiva.
- **CatBoostClassifier**: Similar ao LightGBM em estrutura, mas com implementação própria. Também retorna lista de arrays e necessita tratamento específico para *feature names*.

A solução implementada envolveu a criação de um sistema de detecção automática do tipo de modelo e aplicação da estratégia apropriada para cada caso. O código desenvolvido detecta automaticamente se os `shap_values` são uma lista (LightGBM/-CatBoost) ou um array numpy (scikit-learn) e aplica a lógica correta:

```
# Ajuste automatico do formato de dados
if isinstance(shap_values_single, list):
    # LightGBM e CatBoost: lista de arrays
    shap_values_to_use = shap_values_single[1] if len(
        shap_values_single) > 1 else shap_values_single[0]
else:
    if shap_values_single.ndim == 3:
        # scikit-learn: array 3D (1, n_features, n_classes)
        shap_values_to_use = shap_values_single[..., 1]
    else:
        # Caso simples
        shap_values_to_use = shap_values_single
```

Este sistema garante que todos os modelos sejam explicáveis de forma consistente, independentemente de suas estruturas internas, e que os feature names sejam exibidos corretamente nos gráficos SHAP force plot.

Lições Aprendidas

A integração de técnicas de interpretabilidade com aprendizado de máquina automatizado trouxe uma série de lições importantes, entre as quais destacam-se:

- O controle de versões é essencial para a reprodutibilidade em ambientes com bibliotecas em rápida evolução.
- As técnicas de explicação devem ser compatíveis com a estrutura interna dos modelos e as versões das APIs.
- Sistemas de rastreabilidade devem ser projetados com intencionalidade e estrutura organizada.
- Abordagens probabilísticas e métodos aproximativos são fundamentais para viabilizar explicações em larga escala.
- A compatibilidade entre formatos de dados e modelos requer lógica de detecção e tratamento específica.
- A definição explícita de nomes de atributos é fundamental para garantir a interpretabilidade dos gráficos.
- Mecanismos de fallback e validações robustas são indispensáveis em sistemas voltados à explicabilidade.

Esses aprendizados reforçam a complexidade e os cuidados necessários ao se integrar técnicas de interpretabilidade com ferramentas de AutoML, sobretudo considerando o dinamismo das bibliotecas utilizadas.

Referências Bibliográficas

Alaa, A. e Schaar, M. (2018). Autoprognosis: Automated clinical prognostic modeling via bayesian optimization with structured kernel learning. Em *International conference on machine learning*, pgs. 139–148. PMLR.

Alazab, M. (2020). Automated malware detection in mobile app stores based on robust feature generation. *Electronics*.

Amirian, M., Tuggener, L., Chavarriaga, R., Satyawan, Y. P., Schilling, F.-P., Schwenker, F., e Stadelmann, T. (2021). Two to trust: Automl for safe modelling and interpretable deep learning for robustness. Em *Trustworthy AI-Integrating Learning, Optimization and Reasoning: First International Workshop, TAILOR 2020, Virtual Event, September 4–5, 2020, Revised Selected Papers 1*, pgs. 268–275. Springer.

Arrieta, A. B., Díaz-Rodríguez, N., Del Ser, J., Bennetot, A., Tabik, S., Barbado, A., García, S., Gil-López, S., Molina, D., Benjamins, R., et al. (2020). Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai. *Information fusion*.

Assolin, J., Canto, G., Kreutz, D., e Feitosa, E. (2024). Mh-automl: Transparência, interpretabilidade e desempenho na detecçao de malware android. Em *Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais (SBSeg)*, pgs. 113–120. SBC.

Assolin, J., Kreutz, D., Siqueira, G., Rocha, V., Miers, C., Mansilha, R., e Feitosa, E. (2022). DroidAutoML: uma ferramenta de AutoML para o domínio de detecção de malwares android. Em *Anais Estendidos do XXII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, pgs. 135–142. SBC.

Assolin, J. et. al. (2024). MH-AutoML. https://github.com/SBSegSF24/MH-AutoML.

Bahri, M., Salutari, F., Putina, A., e Sozio, M. (2022). AutoML: state of the art with a focus on anomaly detection, challenges, and research directions. *International Journal of Data Science and Analytics*.

Bifarin, O. O. e Fernández, F. M. (2024). Automated machine learning and explainable ai (automl-xai) for metabolomics: improving cancer diagnostics. *Journal of the American Society for Mass Spectrometry*, 35(6):1089–1100.

Bragança, H., Rocha, V., Assolin, J., Kreutz, D., e Feitosa, E. (2024). Mh-1m: One of the most comprehensive and up-to-date dataset for advanced android malware detection. Em *Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais (SBSeg)*, pgs. 843–849. SBC.

Bragança, H., Rocha, V., Barcellos, L., Souto, E., Kreutz, D., e Feitosa, E. (2023). Capturing the behavior of android malware with mh-100k: A novel and multidimensional dataset. Em *Anais do XXIII Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais*, pgs. 510–515, Porto Alegre, RS, Brasil. SBC.

Brännström, M. (2023). Transparency of complex systems: The semantic transparency framework.

Breiman, L. (2001). Random forests. *Machine learning*.

Cai, L., Li, Y., e Xiong, Z. (2021). Jowmdroid: Android malware detection based on feature weighting with joint optimization of weight-mapping and classifier parameters. *Computers & Security*.

Cao, C., Chicco, D., e Hoffman, M. M. (2020). The mcc-f1 curve: a performance evaluation technique for binary classification. *arXiv preprint arXiv:2006.11278*.

Carvalho, D. V., Pereira, E. M., e Cardoso, J. S. (2019). Machine learning interpretability: A survey on methods and metrics. *Electronics*.

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

Chicco, D. e Jurman, G. (2020). The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation. *BMC genomics*, 21:1–13.

Chicco, D., Warrens, M. J., e Jurman, G. (2021). The matthews correlation coefficient (mcc) is more informative than cohen's kappa and brier score in binary classification assessment. *Ieee Access*, 9:78368–78381.

Colaco, C. W., Bagwe, M. D., Bose, S. A., e Jain, K. (2021). DefenseDroid: A Modern Approach to Android Malware Detection. *Strad Research*, 8(5):271–282.

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

da Silva, M. C., Licari, B., Tavares, G. M., e Junior, S. B. (2024). Benchmarking automl clustering frameworks. Em *AutoML Conference 2024 (ABCD Track)*.

de Mello Assolin, J., Bragança, H., Kreutz, D., Feitosa, E., Saboia, S., Leão, L., e Rocha, V. (2024). Mh-api: Uma solução escalável para detecção de malwares em android. Em *Escola Regional de Redes de Computadores (ERRC)*, pgs. 154–159. SBC.

Domingos, P. (2012). A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87.

Elkan, C. (2001). The foundations of cost-sensitive learning. *International joint conference on artificial intelligence*, 17(1):973–978.

Erickson, N., Mueller, J., Shirkov, A., Zhang, H., Larroy, P., Li, M., e Smola, A. (2020). AutoGluon-Tabular: Robust and accurate AutoML for structured data. https://arxiv.org/abs/2003.06505.

Escovedo, T. e Koshiyama, A. (2022). Bias mitigation in automated machine learning. *IEEE Transactions on Artificial Intelligence*, 3(4):582–596.

Fernández-Delgado, M., Cernadas, E., Barro, S., e Amorim, D. (2014). Do we need hundreds of classifiers to solve real world classification problems? *The Journal of Machine Learning Research*, 15(1):3133–3181.

Ferreira, L., Pilastri, A., Martins, C. M., Pires, P. M., e Cortez, P. (2021). A comparison of AutoML tools for machine learning, deep learning and xgboost. Em *IJCNN*, pgs. 1–8.

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., e Hutter, F. (2015a). Efficient and robust automated machine learning. Em *Advances in Neural Information Processing Systems 28*, pgs. 2962–2970. Curran Associates, Inc.

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., e Hutter, F. (2015b). Efficient and robust automated machine learning. *Advances in neural information processing systems*, 28.

Feurer, M., Springenberg, J. T., e Hutter, F. (2015c). Initializing bayesian hyperparameter optimization via meta-learning.

for Review, A. (20X). Anonymized for review. *Anonymized for Review*.

Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pgs. 1189–1232.

Garg, S., Pundir, P., Rathee, G., Gupta, P., Garg, S., e Ahlawat, S. (2021). On continuous integration/continuous delivery for automated deployment of machine learning models using mlops. Em *2021 IEEE fourth international conference on artificial intelligence and knowledge engineering (AIKE)*, pgs. 25–28. IEEE.

Garouani, M. e Bouneffa, M. (2023). Unlocking the black box: Towards interactive explainable automated machine learning. Em *International Conference on Intelligent Data Engineering and Automated Learning*, pgs. 458–469. Springer.

Gijsbers, P., Bueno, M. L., Coors, S., LeDell, E., Poirier, S., Thomas, J., Bischl, B., e Vanschoren, J. (2024). Amlb: an automl benchmark. *Journal of Machine Learning Research*.

Guerra-Manzanares, A., Bahsi, H., e Nõmm, S. (2021). KronoDroid: Time-based Hybrid-featured Dataset for Effective Android Malware Detection and Characterization. *Computers & Security*, 110:102399.

Gunning, D. e Aha, D. (2019). Darpa's explainable artificial intelligence (xai) program. *AI magazine*, 40(2):44–58.

Guyon, I., Weston, J., Barnhill, S., e Vapnik, V. (2002). Gene selection for cancer classification using support vector machines. *Machine learning*, 46(1-3):389–422.

Han, H., Wang, W.-Y., e Mao, B.-H. (2005). Borderline-smote: a new over-sampling method in imbalanced data sets learning. *Advances in intelligent computing*, pgs. 878–887.

Hasan, R., Dattana, V., Mahmood, S., e Hussain, S. (2024). Towards transparent diabetes prediction: Combining automl and explainable ai for improved clinical insights. *Information*, 16(1):7.

He, H., Bai, Y., Garcia, E. A., e Li, S. (2008). Adasyn: Adaptive synthetic sampling approach for imbalanced learning. *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, pgs. 1322–1328.

He, H. e Garcia, E. A. (2009). Learning from imbalanced data. *IEEE Transactions on knowledge and data engineering*, 21(9):1263–1284.

He, X., Zhao, Y., e Chu, X. (2023). Automl: A systematic review of the state-of-the-art. *ACM Computing Surveys*, 55(8):1–36.

Hinton, G. E. e Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507.

James, G., Witten, D., Hastie, T., e Tibshirani, R. (2013). *An introduction to statistical learning*, volume 112. Springer.

Jian Yang, Xuefeng Li, H. W. (2020). HyperGBM: A Full Pipeline AutoML Tool Integrated With Various GBM Models. https://github.com/DataCanvasIO/HyperGBM. Version 0.2.x.

Jolliffe, I. T. e Cadima, J. (2016). *Principal component analysis*. Springer.

Jurman, G., Riccadonna, S., e Furlanello, C. (2012). A comparison of mcc and cen error measures in multi-class prediction.

Karmaker S. et. al. (2021). Automl to date and beyond: Challenges and opportunities. *ACM Computing Surveys*, 54(8).

Kaspersky (2023). Malwares do google play atingem mais de 600 milhões de downloads em 2023. Accessed: 2025-04-21.

Kennedy, J. e Eberhart, R. (1995). Particle swarm optimization. 4:1942–1948.

Kovalevsky, V., Delhibabu, R., e Zhukova, N. (2024). Automl framework for physical activities recognition. Em *2024 3rd International Conference on Artificial Intelligence For Internet of Things (AIIoT)*, pgs. 1–5. IEEE.

Kreuzberger, D., Kühl, N., e Hirschl, S. (2023). Machine learning operations (mlops): Overview, definition, and architecture. *IEEE Access*.

Kriegel, H.-P., Kröger, P., e Zimek, A. (2010). Outlier detection techniques. *Tutorial at KDD*, 10:1–76.

Kundu, P. P., Anatharaman, L., e Truong-Huu, T. (2021). An empirical evaluation of automated machine learning techniques for malware detection. Em *Proceedings of the 2021 ACM Workshop on Security and Privacy Analytics*.

Le, T. T., Fu, W., e Moore, J. H. (2020). Scaling tree-based automated machine learning to biomedical big data with a feature set selector. *Bioinformatics*, 36(1):250–256.

LeCun, Y., Bengio, Y., e Hinton, G. (2015). Deep learning. *nature*, 521(7553):436–444.

LeDell, E. e Poirier, S. (2020a). H2O AutoML: Scalable automatic machine learning. *7th ICML Workshop on Automated Machine Learning (AutoML)*.

LeDell, E. e Poirier, S. (2020b). H2O AutoML: Scalable automatic machine learning. Em *Proceedings of the AutoML Workshop at ICML*, volume 2020.

Liu, X.-Y., Wu, J., e Zhou, Z.-H. (2009). Exploratory undersampling for class-imbalance learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 39(2):539–550.

Love, P. E., Fang, W., Matthews, J., Porter, S., Luo, H., e Ding, L. (2023). Explainable artificial intelligence (xai): Precepts, models, and opportunities for research in construction. *Advanced Engineering Informatics*.

Lundberg, S. M. e Lee, S.-I. (2017). A unified approach to interpreting model predictions. *Advances in neural information processing systems*.

Maaten, L. v. d. e Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605.

Mahindru, A. (2018). Android permission dataset. *Mendeley Data*.

Mahmood, S., Hasan, R., Hussain, S., e Adhikari, R. (2025). An interpretable and generalizable machine learning model for predicting asthma outcomes: Integrating automl and explainable ai techniques. *World*, 6(1):15.

Martín, A., Calleja, A., Menéndez, H. D., Tapiador, J., e Camacho, D. (2016). Adroit: Android malware detection using meta-information. Em *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pgs. 1–8. IEEE.

McInnes, L., Healy, J., e Melville, J. (2018). Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*.

Mi, J.-X., Li, A.-D., e Zhou, L.-F. (2020). Review study of interpretation methods for future interpretable machine learning. *IEEE Access*.

Mitchell, T. M. (1997). Machine learning.

Moayeri, M., Rabbat, M., Ibrahim, M., e Bouchacourt, D. (2024). Embracing diversity: Interpretable zero-shot classification beyond one vector per class. Em *Proceedings of the 2024 ACM Conference on Fairness, Accountability, and Transparency*, pgs. 2302–2321.

Molino, P., Dudin, Y., e Miryala, S. S. (2019). Ludwig: a type-based declarative deep learning toolbox. *arXiv preprint arXiv:1909.07930*.

Narkar, S., Zhang, Y., Liao, Q. V., Wang, D., e Weisz, J. D. (2021). Model lineupper: Supporting interactive model comparison at multiple levels for automl. Em *Proceedings of the 26th International Conference on Intelligent User Interfaces*, pgs. 170–174.

Naser, M. (2023). Machine learning for all! benchmarking automated, explainable, and coding-free platforms on civil and environmental engineering problems. *Journal of Infrastructure Intelligence and Resilience*.

Nasimian, A. et. al. (2024). Alphaml: A clear, legible, explainable, transparent, and elucidative binary classification platform for tabular data. *Patterns*, 5(1).

Neto, H. A., Alves, R. C., e Campos, S. V. (2020). Nasirt: Automl based learning with instance-level complexity information. *arXiv preprint arXiv:2008.11846*.

Oliveira, S. d., Topsakal, O., e Toker, O. (2024). Benchmarking automated machine learning (automl) frameworks for object detection. *Information*, 15(1):63.

Olson, R. S. e Moore, J. H. (2016). TPOT: A tree-based pipeline optimization tool for automating machine learning. Em *Workshop on automatic machine learning*.

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. *the Journal of machine Learning research*, 12:2825–2830.

Płońska, A. e Płoński, P. (2021). Mljar: State-of-the-art automated machine learning framework for tabular data. *Version 0.10*, 3.

Probst, P., Boulesteix, A.-L., e Bischl, B. (2019). Tunability: Importance of hyperparameters of machine learning algorithms. *The Journal of Machine Learning Research*, 20(1):1934–1965.

Ribeiro, M. T., Singh, S., e Guestrin, C. (2016). "why should i trust you?"explaining the predictions of any classifier. Em *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*.

Rocha, V., Assolin, J., Bragança, H., Kreutz, D., e Feitosa, E. (2023). Amgenerator e amexplorer: Geraçao de metadados e construçao de datasets android. Em *Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais (SBSeg)*, pgs. 41–48. SBC.

Rocha, V., Kreutz, D., Bragança, H., Assolin, J., Pinto, N., e Feitosa, E. (2024). Uma análise compreensiva e exaustiva de métodos de seleção de características para detecção de malware android. Em *Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais (SBSeg)*, pgs. 646–661. SBC.

Singh, A., Patel, S., Bhadani, V., Kumar, V., e Gaurav, K. (2024). Automl-gwl: automated machine learning model for the prediction of groundwater level. *Engineering Applications of Artificial Intelligence*, 127:107405.

Siqueira, G., Kreutz, D., Assolin, J., Costa, E., Miers, C., Mansilha, R., Pontes, J., e Feitosa, E. (2022). Avaliaçao de ferramentas de automl em datasets de detecçao de malwares android. Em *Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais (SBSeg)*, pgs. 302–315. SBC.

Siqueira, G., Rodrigues, G., Kreutz, D., e Feitosa, E. (2021). QuickAutoML: Uma ferramenta para treinamento automatizado de modelos de aprendizado de máquina. Em *WRSeg21*.

Sırt, M. e Eyüpoğlu, C. (2025). Comprehensive benchmarking analysis for evaluating effectiveness of transfer learning-based feature engineering in automl. *The Journal of Cognitive Systems*, 9(2):30–37.

SISTO, A. (2012). Androcrawl: studying alternative android marketplaces.

Snoek, J., Larochelle, H., e Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25.

Soares, T., Kreutz, D., Rocha, V., Costa, E., Leao, L., Pontes, J., Assolin, J., Rodrigues, G., e Feitosa, E. (2022). Uma análise de métodos de seleção de características aplicados à detecção de malwares android. Em *Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais (SBSeg)*, pgs. 288–301. SBC.

Soares, T., Siqueira, G., Barcellos, L., Sayyed, R., Vargas, L., Rodrigues, G., Assolin, J., Pontes, J., Feitosa, E., e Kreutz, D. (2021). Detecção de Malwares Android: datasets e reprodutibilidade. Em *VI Workshop Regional de Segurança da Informação e de Sistemas Computacionais (WRSeg)*, Charqueadas-RS, Brasil.

Statista (2024). Most downloaded mobile apps from the google play store worldwide in march 2024. Accessed: 2025-04-21.

Statista (2025). Number of android users, apps, and downloads worldwide as of 2024. Accessed: April 14, 2025.

Sun, L., Li, Z., Yan, Q., Srisa-an, W., e Pan, Y. (2016). Sigpid: significant permission identification for android malware detection. Em *2016 11th International Conference on Malicious and Unwanted Software (MALWARE)*, pgs. 1–8.

Thornton, C., Hutter, F., Hoos, H. H., e Leyton-Brown, K. (2013). Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. Em *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pgs. 847–855.

Tian, J. e Che, C. (2024). Automated machine learning: A survey of tools and techniques. *Journal of Industrial Engineering and Applied Science*, 2(6):71–76.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288.

Trirat, P., Jeong, W., e Hwang, S. J. (2024). Automl-agent: A multi-agent llm framework for full-pipeline automl. *arXiv preprint arXiv:2410.02958*.

Truong, A., Walters, A., Goodsitt, J., Hines, K., Bruss, C. B., e Farivar, R. (2019). Towards automated machine learning: Evaluation and comparison of AutoML approaches and tools. Em *IEEE 31st ICTAI*.

Tsamardinos, I., Charonyktakis, P., Lakiotaki, K., Borboudakis, G., Zenklusen, J. C., Juhl, H., Chatzaki, E., e Lagani, V. (2020). Just add data: Automated predictive modeling and biosignature discovery. *BioRxiv*.

Tschiedel, L. C., Rocha, V., Kreutz, D., Bragança, H., Quincozes, S. E., Nogueira, A. G., e Assolin, J. (2024). Sigapi autocraft: uma ferramenta de seleção de características com capacidade de generalização. Em *Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais (SBSeg)*, pgs. 169–176. SBC.

Vakhrushev, A., Ryzhkov, A., Savchenko, M., Simakov, D., Damdinov, R., e Tuzhilin, A. (2021). Lightautoml: Automl solution for a large financial services ecosystem. *arXiv preprint arXiv:2109.01528*.

Vilanova, L., Kreutz, D., Assolin, J., Quincozes, V., Miers, C., Mansilha, R., e Feitosa, E. (2022). Adbuilder: uma ferramenta de construçao de datasets para detecçao de malwares android. Em *Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais (SBSeg)*, pgs. 143–150. SBC.

Waring, J., Lindvall, C., e Umeton, R. (2020). Automated machine learning: Review of the state-of-the-art and opportunities for healthcare. *Artificial intelligence in medicine*, 104:101822.

Weerts, H., Pfisterer, F., Feurer, M., Eggensperger, K., Bergman, E., Awad, N., Vanschoren, J., Pechenizkiy, M., Bischl, B., e Hutter, F. (2023). Can fairness be automated? guidelines and opportunities for fairness-aware AutoML.

Wever, M., Tornede, A., Mohr, F., e Hüllermeier, E. (2021). Automl for multi-label classification: Overview and empirical evaluation. *IEEE transactions on pattern analysis and machine intelligence*, 43(9):3037–3054.

Ye, T., Meng, J., Xiao, Y., Lu, Y., Zheng, A., e Liang, B. (2025). Integrated automl-based framework for optimizing shale gas production: A case study of the fuling shale gas field. *Energy Geoscience*, 6(1):100365.

Yerima, S. Y. e Sezer, S. (2018). Droidfusion: A novel multilevel classifier fusion approach for android malware detection. *IEEE transactions on cybernetics*, 49(2):453–466.

Yu, L. e Liu, H. (2003). Feature selection for high-dimensional data: A fast correlation-based filter solution. *Proceedings of the 20th international conference on machine learning (ICML-03)*, pgs. 856–863.

Yuan, H., Yu, K., Xie, F., Liu, M., e Sun, S. (2024). Automated machine learning with interpretation: a systematic review of methodologies and applications in healthcare. *Medicine Advances*, 2(3):205–237.

Zimmer, L., Lindauer, M., e Hutter, F. (2020). Auto-pytorch tabular: Multi-fidelity metalearning for efficient and robust autodll. arxiv 2020. *arXiv preprint arXiv:2006.13799*.

Zimmer, L., Lindauer, M., e Hutter, F. (2021). Auto-pytorch: multi-fidelity metalearning for efficient and robust autodl. *IEEE Trans. on Pattern Analysis and MI*, 43(9):3079–3090.

Zöller, M.-A. e Huber, M. F. (2021). Benchmark and survey of automated machine learning frameworks. *Journal of artificial intelligence research*, 70:409–472.

Zöller, M.-A., Titov, W., Schlegel, T., e Huber, M. F. (2023). Xautoml: a visual analytics tool for understanding and validating automated machine learning. *ACM Transactions on Interactive Intelligent Systems*, 13(4):1–39.

Apêndice A: Resumo dos artefatos gerados pela MH-AutoML

.

📋 RESUMO DE ARTEFATOS - MH-AutoML

🧮 Resumo

Este documento apresenta uma análise detalhada dos principais artefatos gerados pelo pipeline MH-AutoML, uma solução desenvolvida para a detecção de malware em aplicações Android. Os resultados e as configurações aqui descritas são baseados em um conjunto de dados de teste, composto por **15.036 amostras e 51 características**.

O dataset reflete um cenário realista de distribuição de classes, onde a proporção de amostras benignas é naturalmente maior que a de malwares, apresentando **63.01% de amostras benignas e 36.99% de amostras maliciosas**. Essa característica do conjunto de dados é fundamental para avaliar a capacidade do modelo em lidar com a assimetria comum em problemas de detecção de ameaças.

Além das visualizações gráficas que ilustram cada etapa do processo, o pipeline também gera arquivos CSV e um modelo serializado, que registram escolhas e resultados cruciais. Ao longo das seções seguintes, serão abordadas as etapas cruciais do processo de Machine Learning:

- **Pré-processamento dos dados:** Com destaque para a análise de valores faltantes.
- **Engenharia de características:** Incluindo a seleção de features via LASSO e ANOVA, e a redução de dimensionalidade com PCA. Além disso, arquivos como `Features_Selected_20250701_232221.csv` e `treino_20250701_232146.csv` fornecem o registro exato das características selecionadas e dos dados utilizados no treinamento.
- **Otimização do modelo:** Detalhando a importância dos hiperparâmetros e a curva de otimização. O arquivo `Hyperparameters_Results.csv` documenta todas as tentativas de otimização e as métricas de desempenho correspondentes, enquanto `Models_Ranking.csv` oferece uma classificação dos modelos testados.
- **Avaliação de desempenho:** Apresentando métricas como matriz de confusão, curvas ROC, Precisão-Recall e avaliação por classe.
- **Interpretabilidade do modelo:** Utilizando técnicas avançadas como SHAP e LIME, além da análise da árvore de decisão, para garantir a transparência e a confiabilidade das

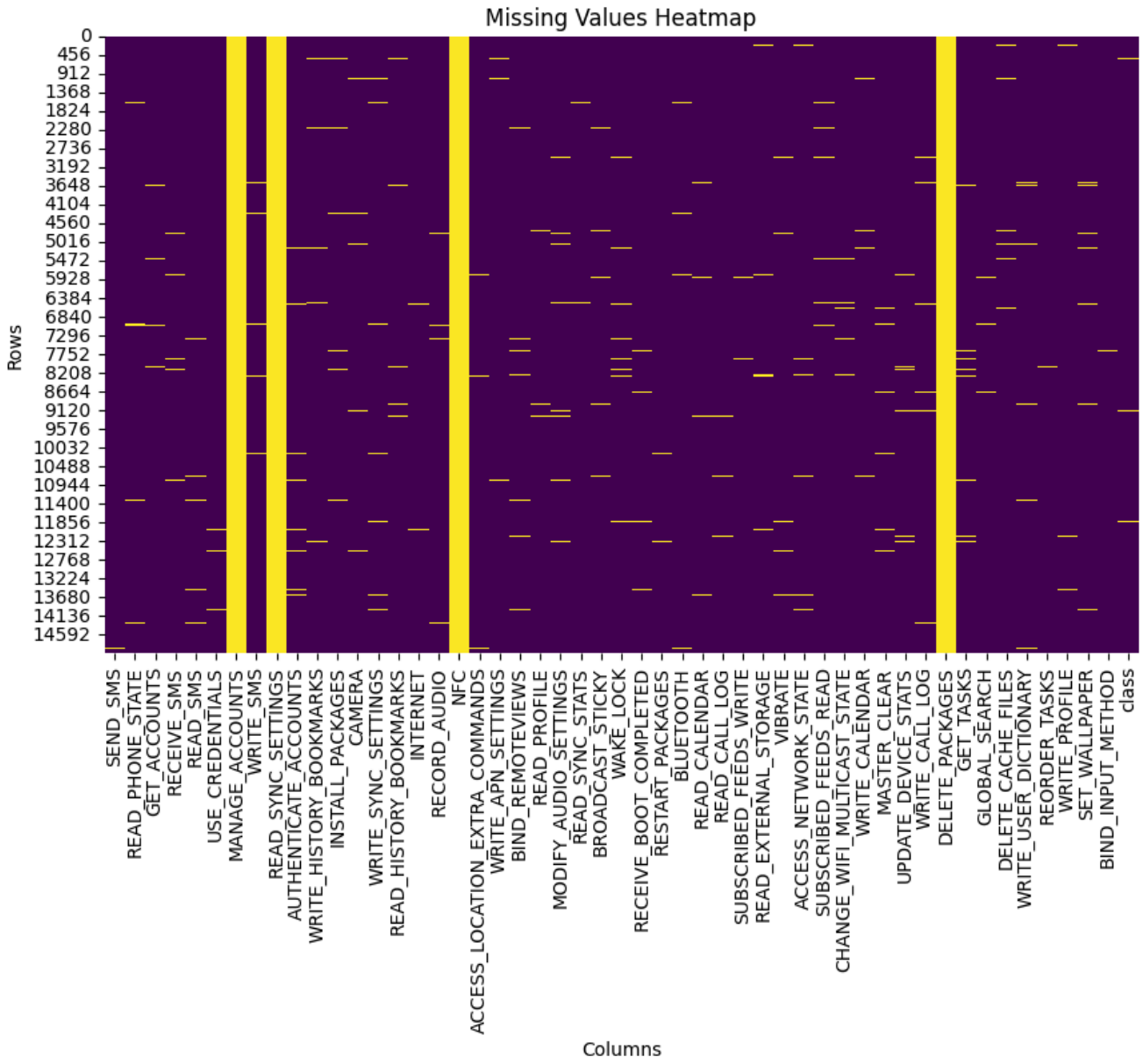
previsões.

- **Performance geral do pipeline:** Avaliando o consumo de tempo e memória RAM por etapa.

Cada artefato visual e estatístico será explicado para oferecer uma compreensão completa do fluxo de trabalho, das escolhas técnicas e dos resultados alcançados pelo sistema MH-AutoML, incluindo o `best_model_20250701_232146.pkl`, que representa o modelo final otimizado.

1. 📊 Pré Processamento

1.2 Análise de Valores Faltantes



Explicação:

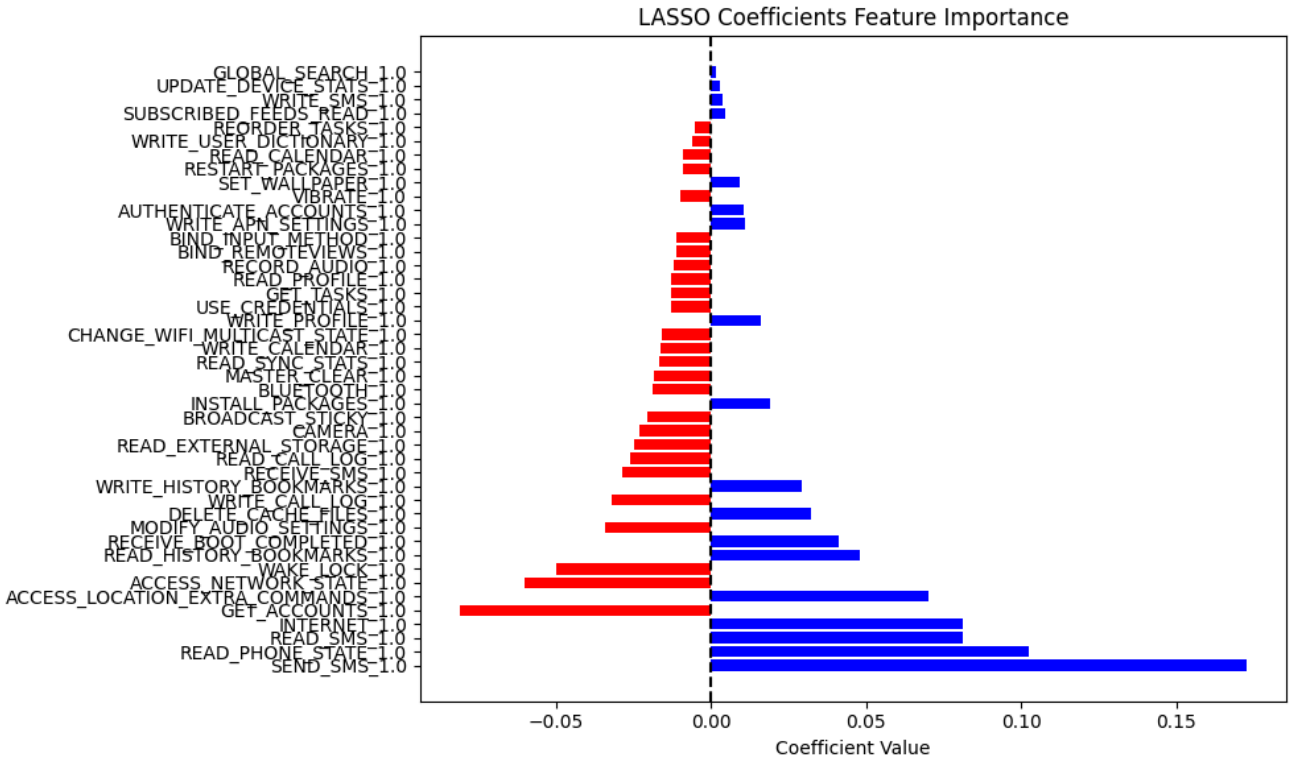
A figura abaixo apresenta um *heatmap* de valores ausentes no conjunto de dados, em que as regiões em roxo indicam dados completos, enquanto as regiões em amarelo representam valores faltantes. Essa visualização é útil para identificar padrões sistemáticos de ausência, indicando que certas características foram coletadas de forma inconsistente entre as amostras.

É possível observar que algumas colunas apresentam uma proporção significativa de valores ausentes, o que pode comprometer a performance de modelos de aprendizado de máquina. Dessa forma:

- **Características com mais de 50% de valores ausentes** devem ser consideradas para remoção;
- **Características com taxa moderada de ausência** podem ser tratadas com técnicas de imputação.

2. ⚙️ Engenharia de Features

2.1 Seleção de Características LASSO e ANOVA



Explicação LASSO:

O gráfico acima apresenta a importância das características extraída a partir dos coeficientes do modelo LASSO (*Least Absolute Shrinkage and Selection Operator*), que aplica regularização L1 durante o treinamento. Nesse contexto:

- As **barras azuis** representam características com coeficientes positivos, que estão positivamente associadas à classe-alvo (por exemplo, “malware”);
- As **barras vermelhas** indicam características com coeficientes negativos, associadas à classe oposta (por exemplo, “benigno”).

O valor absoluto do coeficiente indica o peso da influência da característica no modelo. Características com coeficiente zero foram automaticamente eliminadas pelo LASSO, contribuindo para a seleção de um subconjunto mais relevante de atributos e promovendo a interpretabilidade do modelo.

Destacam-se, por exemplo, permissões como `SEND_SMS`, `READ_PHONE_STATE`, `INTERNET` e `READ_SMS`, que apresentaram os maiores coeficientes positivos, sugerindo forte associação com o comportamento malicioso de aplicações Android. Esse tipo de análise é essencial para entender o papel individual de cada atributo no processo de classificação automatizada.

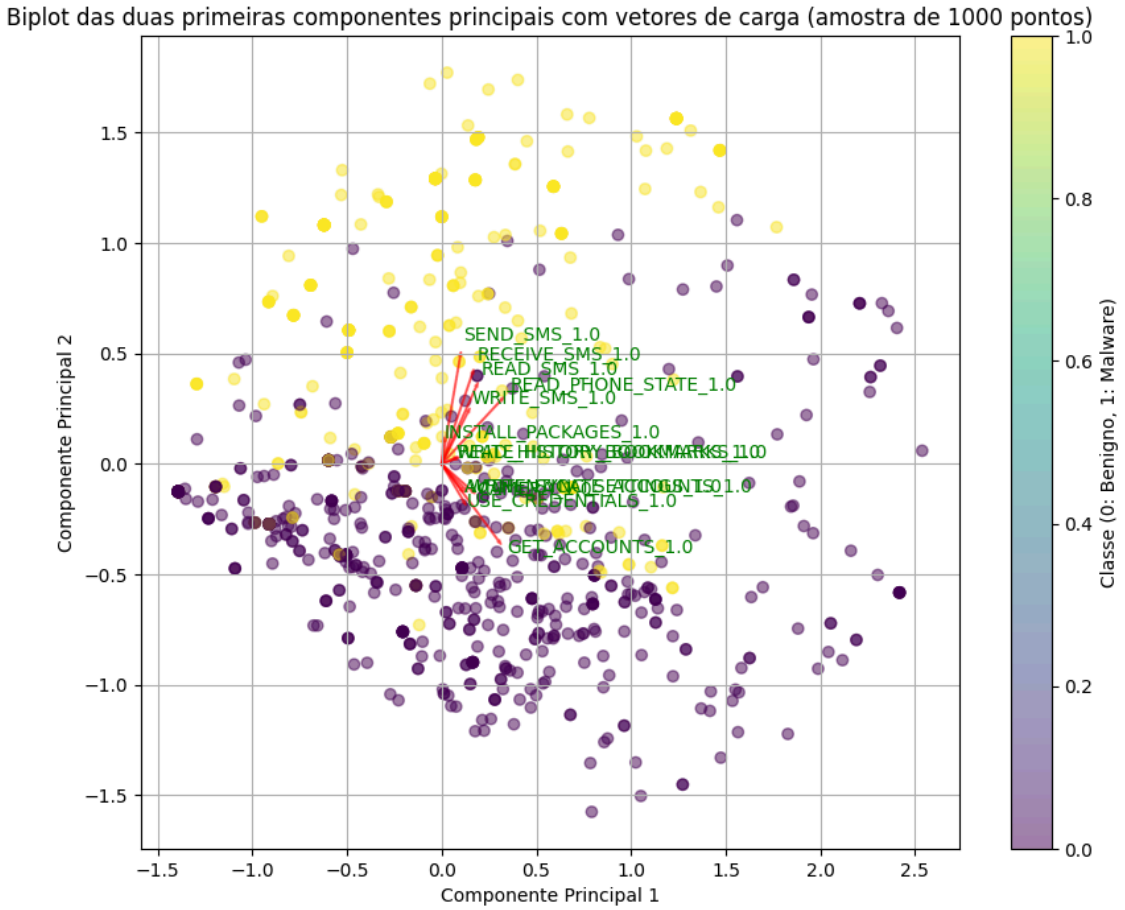
Importância de Características com ANOVA

O gráfico acima representa os *F-values* obtidos pela aplicação do teste estatístico ANOVA (*Analysis of Variance*) para avaliação univariada de importância de cada característica em relação à variável-alvo.

Características com valores de F mais altos possuem maior poder discriminativo entre as classes. Diferente do LASSO, que realiza penalizações e seleção multivariada, a ANOVA avalia cada atributo de forma independente, oferecendo uma visão estatística útil para filtragem inicial de atributos.

Essa abordagem é especialmente útil em etapas preliminares de seleção de características, sendo frequentemente combinada com métodos de regularização ou árvores de decisão para análises mais robustas.

2.2 Redução de Dimensionalidade PCA



Explicação:

O gráfico apresentado é um **Biplot de Análise de Componentes Principais (PCA)**. Este tipo de visualização tem como objetivo reduzir a dimensionalidade de um conjunto de dados complexo (neste caso, as permissões de aplicativos) para duas dimensões (Componente Principal 1 e Componente Principal 2), que capturam a maior parte da variabilidade dos dados.

O gráfico exibe simultaneamente:

1. **As amostras (pontos):** Cada ponto representa um aplicativo, colorido de acordo com sua classe. Roxo (valor 0) indica um aplicativo **Benigno**, e amarelo (valor 1) indica **Malware**.

2. **As variáveis originais (vetores):** As setas vermelhas partindo da origem são os vetores de carga, representando as permissões do sistema (features). A direção e o comprimento de cada vetor indicam como cada permissão influencia os dois componentes principais.

Principais destaques:

A análise do gráfico revela como as permissões (features) se relacionam com as classes de aplicativos (Benigno vs. Malware):

- **Separação de Classes:** Existe uma tendência clara de separação entre as classes. Aplicativos **Malware (amarelo)** estão concentrados predominantemente no quadrante superior direito, indicando que possuem valores positivos tanto para a Componente Principal 1 quanto para a Componente Principal 2. Aplicativos **Benignos (roxo)** estão mais dispersos, mas com uma forte concentração no lado esquerdo do gráfico (valores negativos de Componente Principal 1).

- **Influência das Permissões:** Os vetores de carga nos mostram as permissões mais influentes:

    - **Fortemente associadas a Malware:** Permissões como `SEND_SMS`, `READ_PHONE_STATE`, `INSTALL_PACKAGES` e `WRITE_SMS` têm vetores que apontam para a mesma direção da concentração de malware (direita e para cima). Isso indica que essas permissões são fortes indicadores de um aplicativo malicioso.

    - **Correlação entre Permissões:** Os vetores para `SEND_SMS`, `RECEIVE_SMS`, `READ_SMS` e `WRITE_SMS` estão muito próximos, apontando na mesma direção. Isso sugere que essas permissões são altamente correlacionadas, ou seja, se um aplicativo solicita uma delas, é muito provável que solicite as outras também.

Interpretação:

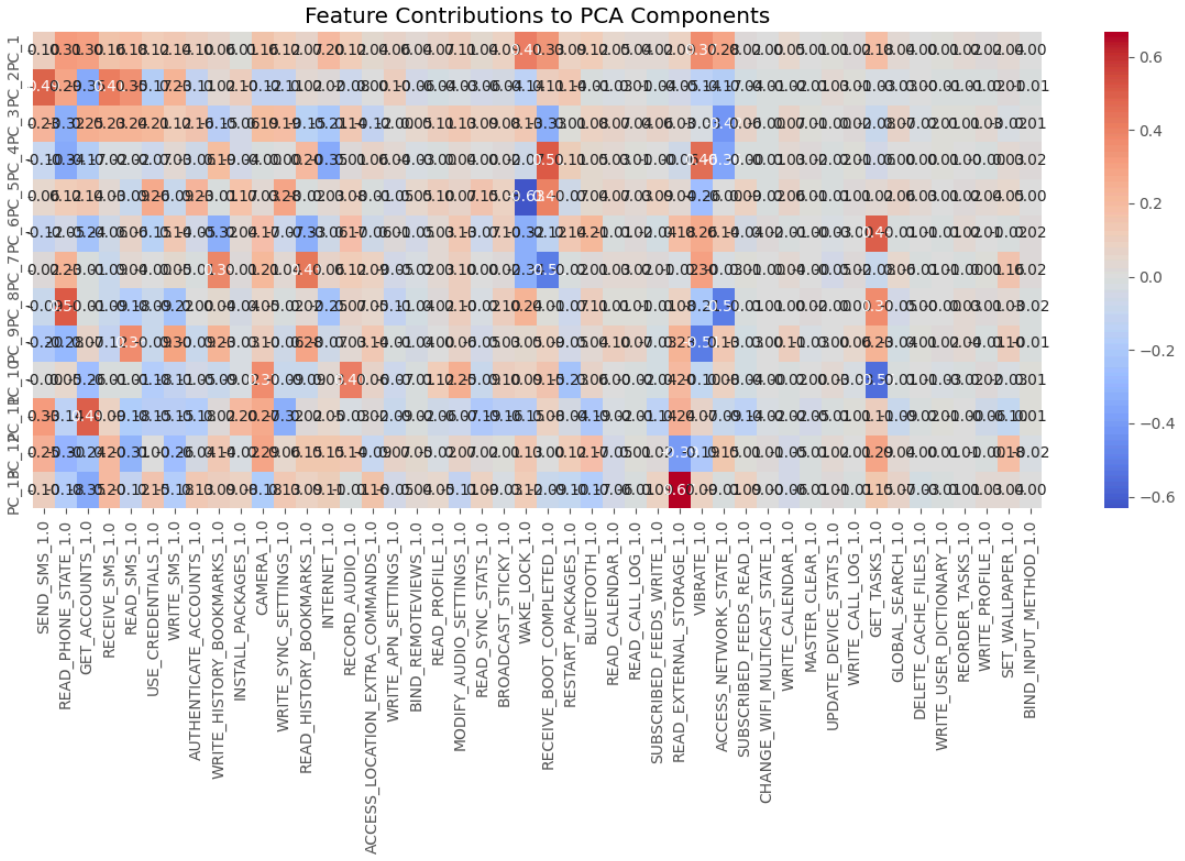
O biplot demonstra que as duas primeiras componentes principais conseguem capturar características importantes que distinguem aplicativos benignos de maliciosos.

A **Componente Principal 1** (eixo horizontal) parece ser um forte diferenciador geral. Valores positivos nesta componente, influenciados principalmente por permissões como `READ_PHONE_STATE` e `INSTALL_PACKAGES`, estão fortemente associados a malware.

A **Componente Principal 2** (eixo vertical), influenciada principalmente pela permissão `SEND_SMS`, ajuda a refinar essa separação, especialmente para o cluster de malware.

A sobreposição entre os pontos roxos e amarelos indica que apenas com estas duas componentes não é possível separar perfeitamente as duas classes, mas a tendência é evidente. O gráfico fornece insights valiosos para a engenharia de features, mostrando que um modelo de machine learning provavelmente atribuirá grande importância a permissões como `SEND_SMS` e `READ_PHONE_STATE` para detectar atividades maliciosas. Em resumo, a visualização confirma que o comportamento de solicitação de certas permissões é um fator crucial para a classificação de malware.

2.2 Mapa de calor contribuição das características PCA



Explicação:

O gráfico apresentado é um **mapa de calor (heatmap)** que detalha as contribuições (ou "cargas") de cada feature original (permissões de aplicativos, listadas no eixo Y) para cada uma das dez primeiras Componentes Principais (PCs), de PC 1 a PC 10 (listadas no eixo X).

Esta visualização nos permite entender a "composição" de cada componente principal. A intensidade e a cor de cada célula indicam a força e a direção da influência de uma feature sobre um componente:

- **Cor Vermelha (valores positivos):** Indica uma contribuição positiva. Quando o valor de uma feature com carga positiva aumenta, o escore do componente principal também tende a aumentar.

- **Cor Azul (valores negativos):** Indica uma contribuição negativa. Quando o valor de uma feature com carga negativa aumenta, o escore do componente principal tende a diminuir.

- **Cor Clara/Branca (valores próximos de zero):** Indica que a feature tem pouca ou nenhuma influência sobre aquele componente específico.

Principais destaques:

O mapa de calor revela que cada componente principal é dominado por um pequeno conjunto de features correlacionadas, representando um tipo específico de comportamento ou funcionalidade do aplicativo.

- **Componente Principal 1 (PC 1):** É fortemente dominada por duas features com cargas positivas muito altas: `ACCESS_NETWORK_STATE_1.0` e `INTERNET_1.0`. Todas as outras features têm uma contribuição quase nula para este componente.
- **Componente Principal 2 (PC 2):** É majoritariamente definida pela permissão `SEND_SMS_1.0`, que possui a maior carga positiva. Outras permissões relacionadas a SMS (`RECEIVE_SMS` e `READ_SMS`) também contribuem positivamente, mas com menor intensidade.
- **Componente Principal 3 (PC 3):** É primariamente influenciada positivamente pela permissão `READ_PHONE_STATE_1.0`.
- **Componente Principal 4 (PC 4):** Possui uma forte carga negativa da feature `USE_CREDENTIALS_1.0`.
- **Outros Componentes:** Cada componente subsequente é definido por outras features. Por exemplo, `PC 5` é influenciado por `WRITE_SETTINGS_1.0`, e `PC 8` é negativamente influenciado por `RESTART_PACKAGES_1.0`.

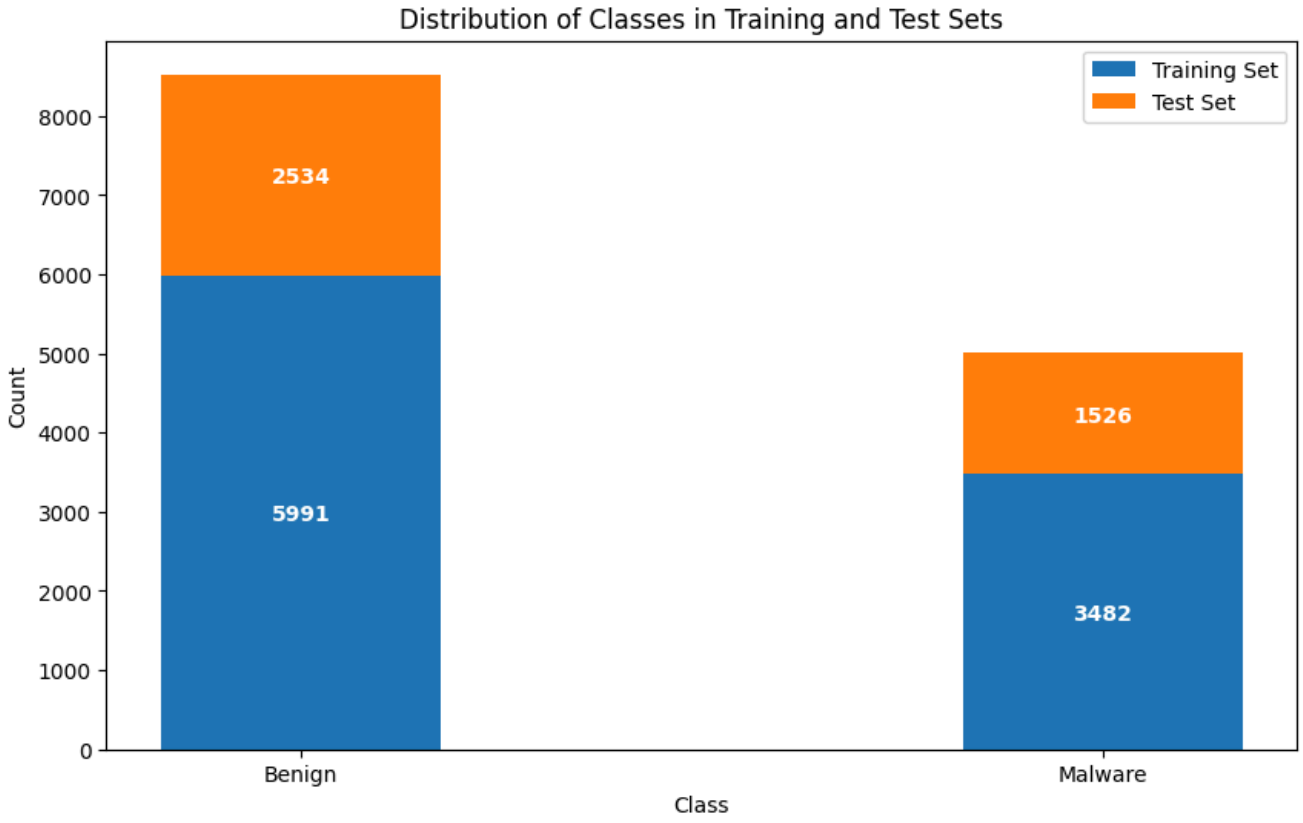
Interpretação:

Este mapa de calor é fundamental para interpretar o que cada componente principal representa em termos práticos. Em vez de analisar dezenas de permissões, podemos entender a variabilidade dos dados através de 10 “conceitos” ou “padrões de comportamento” independentes.

- **PC 1 pode ser interpretado como o “Componente de Acesso à Internet”**. Aplicativos com pontuação alta neste componente são aqueles que fazem uso intensivo da rede.
- **PC 2 representa a “Componente de Funcionalidade SMS”**. Aplicativos que enviam, recebem ou leem SMS terão uma pontuação alta neste componente. (Conectando com a análise anterior, esta era uma característica chave para identificar malware).
- **PC 3 pode ser visto como a “Componente de Identificação do Dispositivo”**, já que `READ_PHONE_STATE` permite o acesso a informações como o IMEI do telefone.
- **PC 4 representa um “Componente de Gerenciamento de Contas/Credenciais”**, e sua carga negativa sugere um comportamento distinto para apps que usam essa permissão.

Em suma, o gráfico “traduz” os componentes matemáticos abstratos (PC 1, PC 2, etc.) em comportamentos de aplicativos compreensíveis. Essa análise é extremamente útil para a engenharia de features e para a interpretabilidade de modelos de machine learning. Por exemplo, se um modelo de classificação identifica que PC 2 é um forte preditor de malware, este mapa de calor nos diz explicitamente que a capacidade de **enviar SMS** é a razão por trás dessa predição.

2.3 Distribuição de Classes



Explicação:
A figura abaixo apresenta a distribuição das classes (*Benign* e *Malware*) nos conjuntos de treinamento e teste. A visualização mostra o número de amostras de cada classe em ambos os conjuntos, permitindo avaliar se a divisão foi balanceada e representativa.

- **Treinamento**:
  - Benign: **5991**
  - Malware: **3482**
  - Total: **9473** amostras
- **Teste**:
  - Benign: **2534**

- Malware: **1526**
- Total: **4060** amostras

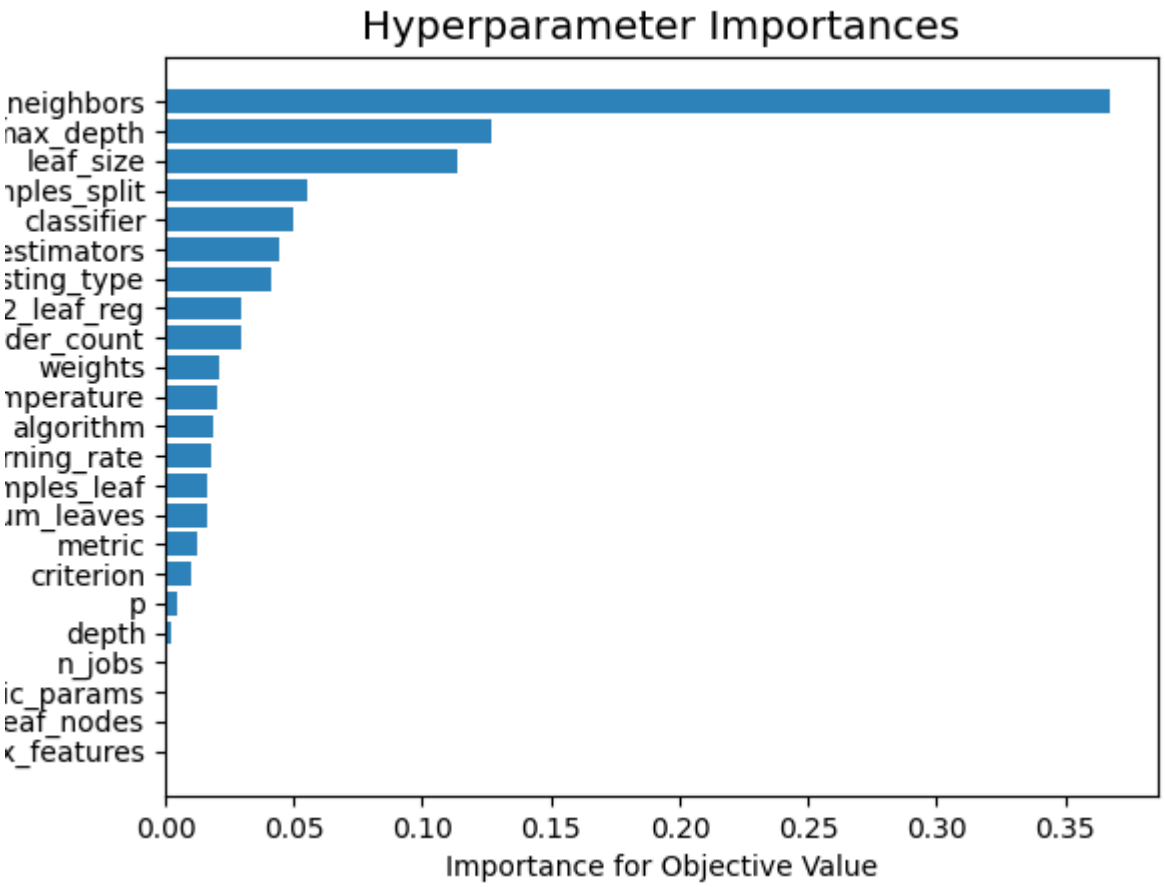
Proporção (Benign:Malware): ~1.7:1 em ambos os conjuntos.

Principais Pontos:
✓ Divisão estratificada (proporção preservada).
✓ Conjunto de treinamento maior para aprendizado adequado.

Essa distribuição é crucial para garantir que o modelo seja treinado e avaliado em dados que refletem a proporção real das classes, evitando viés.

3. 🎯 Otimização de Modelo

3.1 Importância de Hiperparâmetros



Explicação:
O gráfico apresenta a importância relativa dos hiperparâmetros no desempenho do modelo,

conforme avaliado pelo framework Optuna. Os parâmetros são ordenados por sua influência no valor objetivo, onde valores mais altos indicam maior impacto.

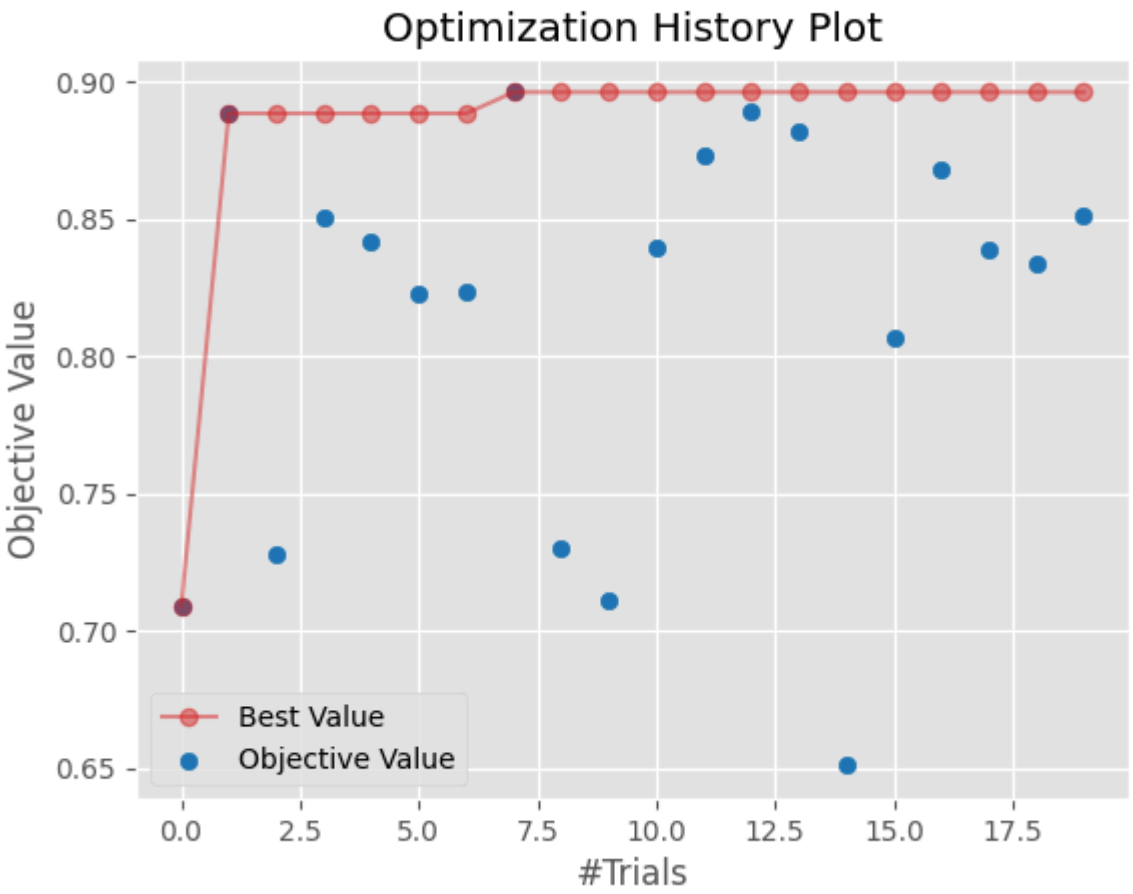
Principais destaques:

Os hiperparâmetros mais significativos são:

- **neighbors** (importância ~0.35) - o parâmetro com maior influência
- **nav_depth** (importância ~0.25)
- **leaf_size** (importância ~0.20)
- **nples_split** (importância ~0.15)
- **classifier** (importância ~0.10)

Interpretação:
Os parâmetros na extremidade direita do gráfico (com importância acima de 0.10) merecem atenção especial durante o ajuste fino do modelo, pois pequenas variações nestes podem impactar significativamente os resultados. Por outro lado, parâmetros com importância próxima de zero (como 'ic_params' e 'eaf_nodes') têm efeito mínimo e podem ser mantidos com valores padrão para simplificar o processo de otimização.
O gráfico de importância dos hiperparâmetros fornece insights sobre quais parâmetros do modelo exercem maior influência no desempenho, permitindo reduzir o espaço de busca e, consequentemente, tornar o processo de otimização mais eficiente.

3.2 Curva de Otimização



Explicação:

O gráfico de histórico de otimização documenta a evolução do desempenho do modelo ao longo de sucessivas tentativas de ajuste de hiperparâmetros, revelando importantes padrões:

Trajetória de Melhoria

- O processo iniciou com um desempenho modesto (0.65), característico de configurações não otimizadas
- Observa-se uma rápida ascensão na fase inicial (0.65 → 0.80 em 5 tentativas), demonstrando a eficácia da abordagem de otimização
- A fase intermediária (tentativas 5-15) apresenta ganhos incrementais, típico de processos de refinamento
- O platô alcançado (0.90) após 15 tentativas sugere a exploração adequada do espaço de parâmetros

Dinâmica de Convergência

1. **Fase Exploratória** (0-5 tentativas): Melhorias rápidas indicam que o algoritmo encontrou rapidamente regiões promissoras no espaço de parâmetros

2. **Fase de Refinamento** (5-15 tentativas): Aperfeiçoamento gradual sugere ajustes finos nas combinações de parâmetros

3. **Fase de Estabilização** (pós-15 tentativas): Manutenção do desempenho máximo indica possível esgotamento das melhorias viáveis

Interpretação Técnica

- A diferença entre os valores inicial (0.65) e final (0.90) representa um ganho de ~38% no desempenho

- A estabilização precoce (em ~15 tentativas) pode sugerir:

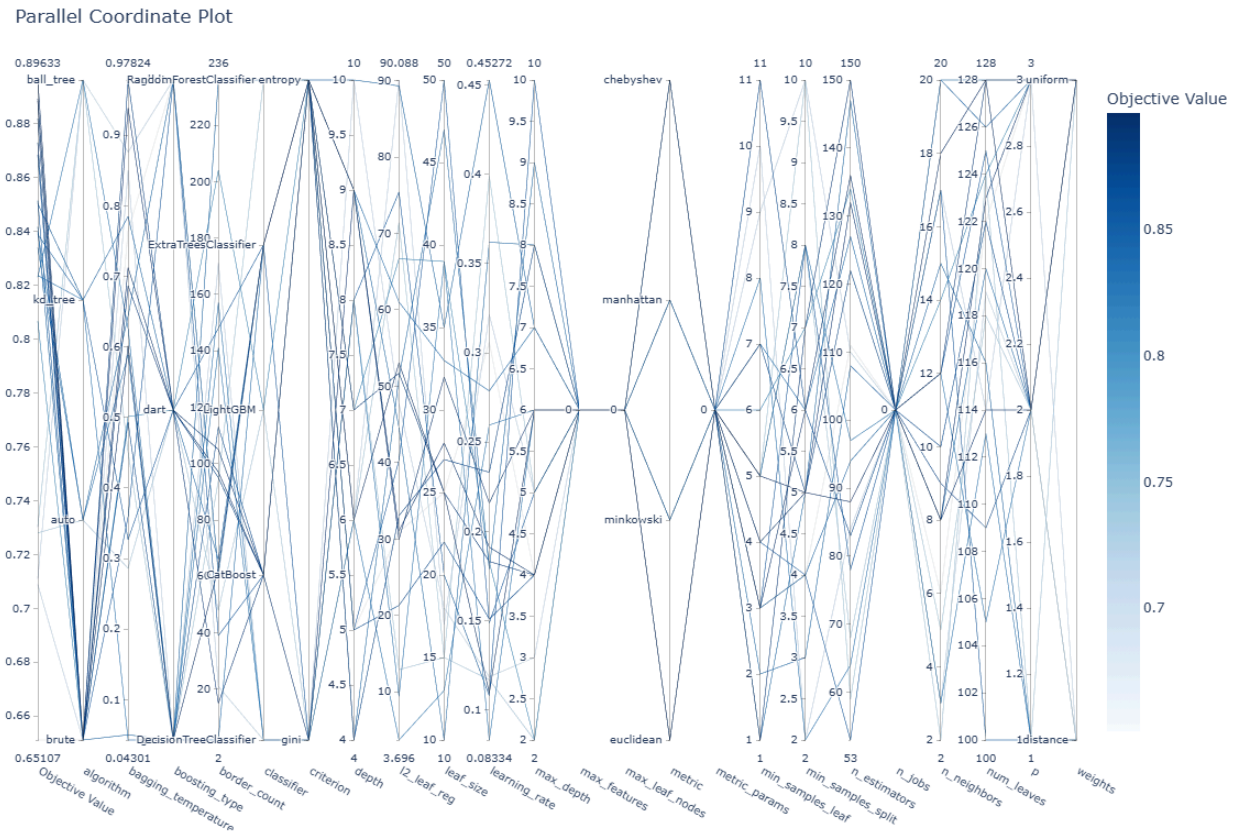
    - Eficiência do algoritmo de otimização

    - Espaço de parâmetros bem definido

    - Potencial para melhorias através da expansão do espaço de busca

Esta análise demonstra a efetividade do processo de otimização, com ganhos significativos de desempenho alcançados de forma eficiente, ao mesmo tempo que aponta oportunidades para refinamentos adicionais.

3.2 Coordenadas Paralelas

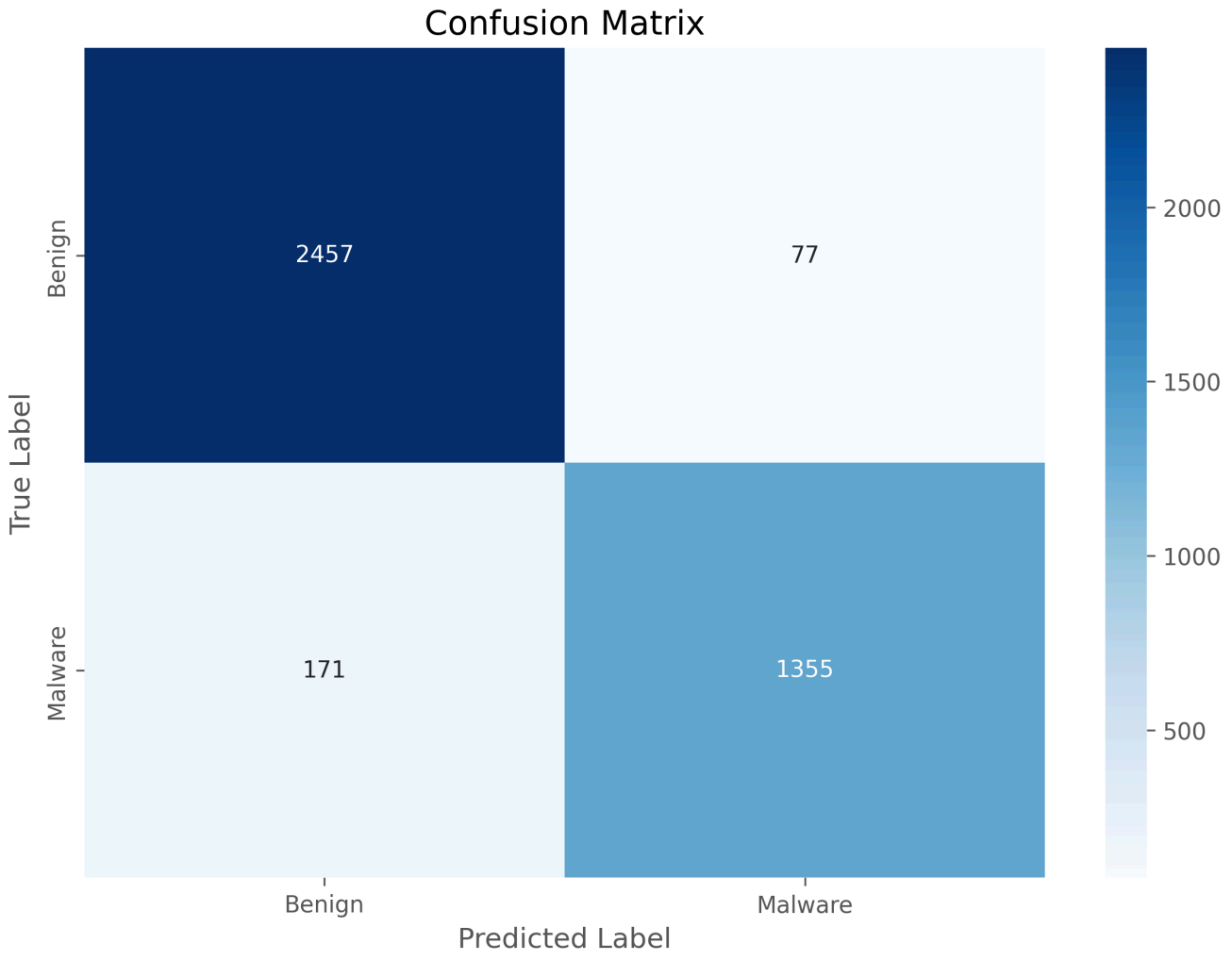


Explicação:

O gráfico de coordenadas paralelas revela as relações entre múltiplos hiperparâmetros e o desempenho do modelo (0.89 a 0.97), destacando combinações ótimas através de padrões visíveis nas linhas superiores - alguns parâmetros mostram forte correlação com bons resultados (valores concentrados em faixas específicas), enquanto outros apresentam variação aleatória, indicando menor influência, sugerindo que a otimização deve focar nos intervalos associados às melhores execuções e reduzir a busca em parâmetros menos impactantes.

4. 📈 Avaliação de Desempenho

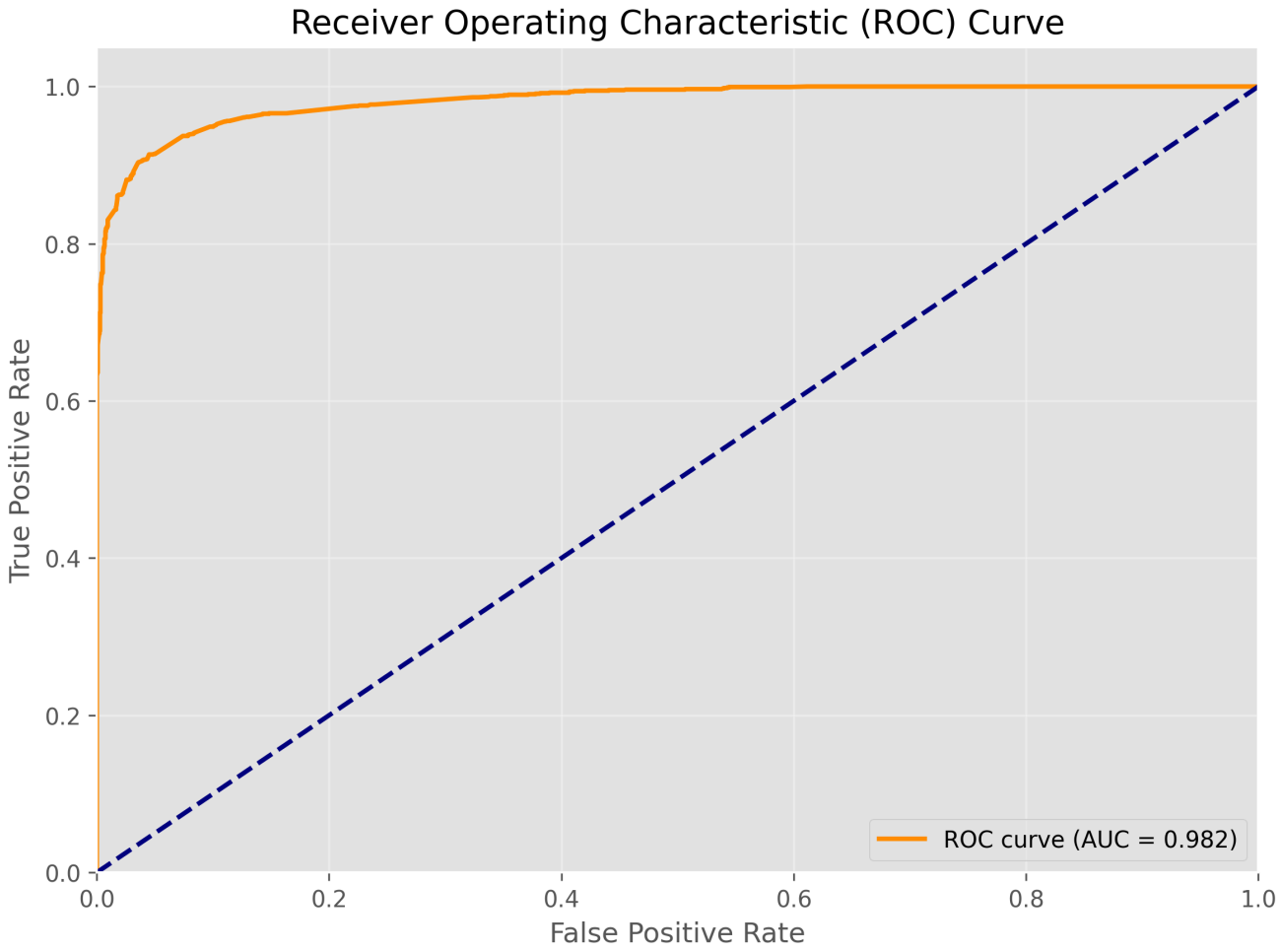
4.1 Matriz de Confusão



Explicação:

A matriz de confusão demonstra a capacidade do modelo em distinguir entre amostras benignas e maliciosas. Das **2.534** amostras benignas reais, **2.457** foram corretamente classificadas, com apenas **77 falsos positivos**, o que demonstra **alta especificidade**.Das **1.526** amostras maliciosas, **1.355** foram corretamente identificadas, com **171 falsos negativos**, revelando uma **boa sensibilidade**, embora ainda haja espaço para melhorias na detecção de ameaças. O desempenho indica que o modelo apresenta **baixo índice de alarmes falsos** e uma **eficácia significativa na identificação de malwares**, sendo adequado para cenários que exigem confiança tanto na detecção quanto na minimização de alertas indevidos.

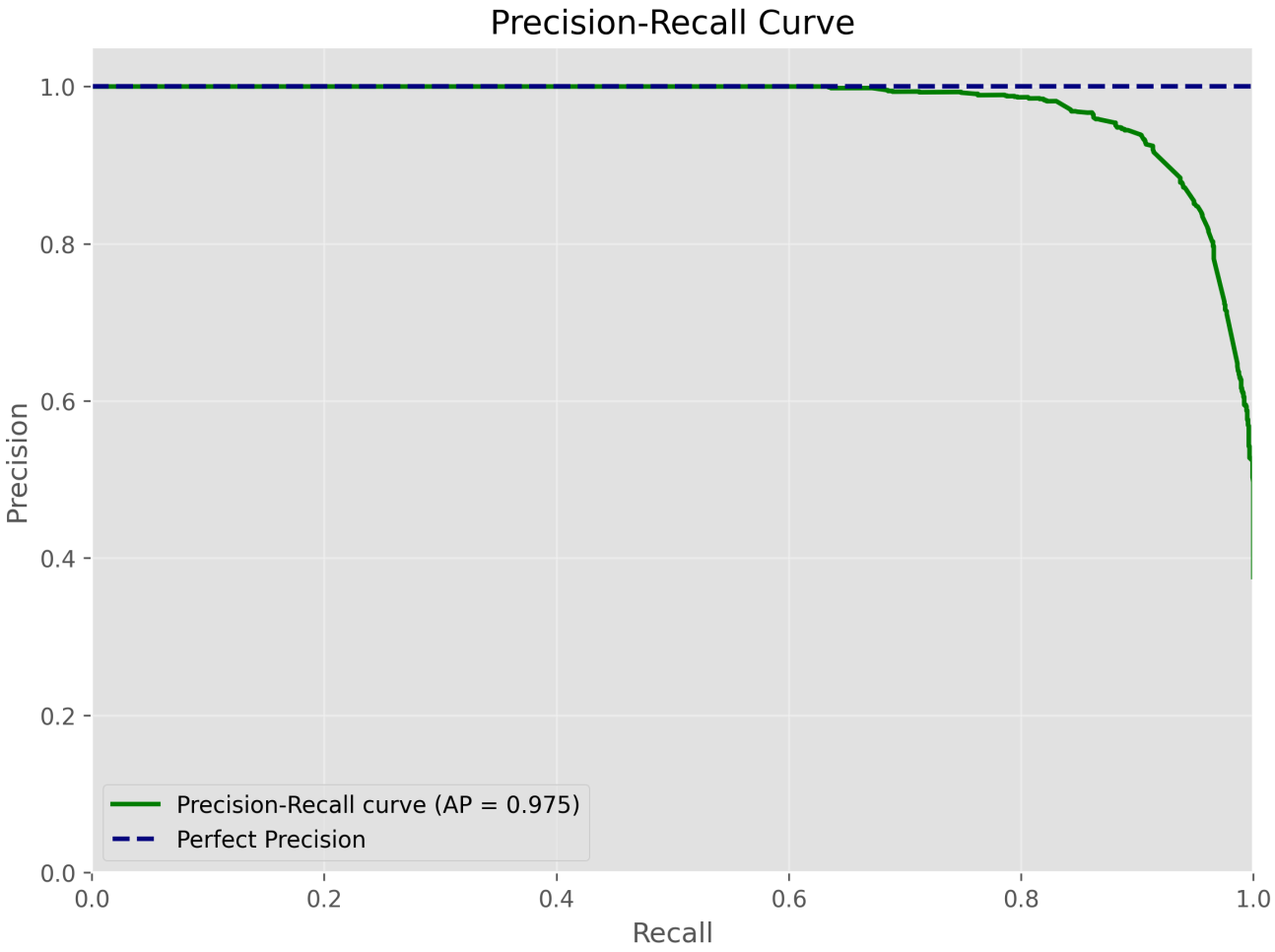
4.2 Curvas de Avaliação AUC-ROC



Explicação:

A curva ROC (Receiver Operating Characteristic) compara a taxa de verdadeiros positivos (sensibilidade) com a taxa de falsos positivos, em diferentes limiares de decisão. A área sob a curva (AUC) indica a capacidade do modelo de distinguir entre as classes. Um valor de AUC próximo de 1, como o obtido (0.982), demonstra excelente desempenho discriminativo, com mínima sobreposição entre classes benignas e maliciosas.

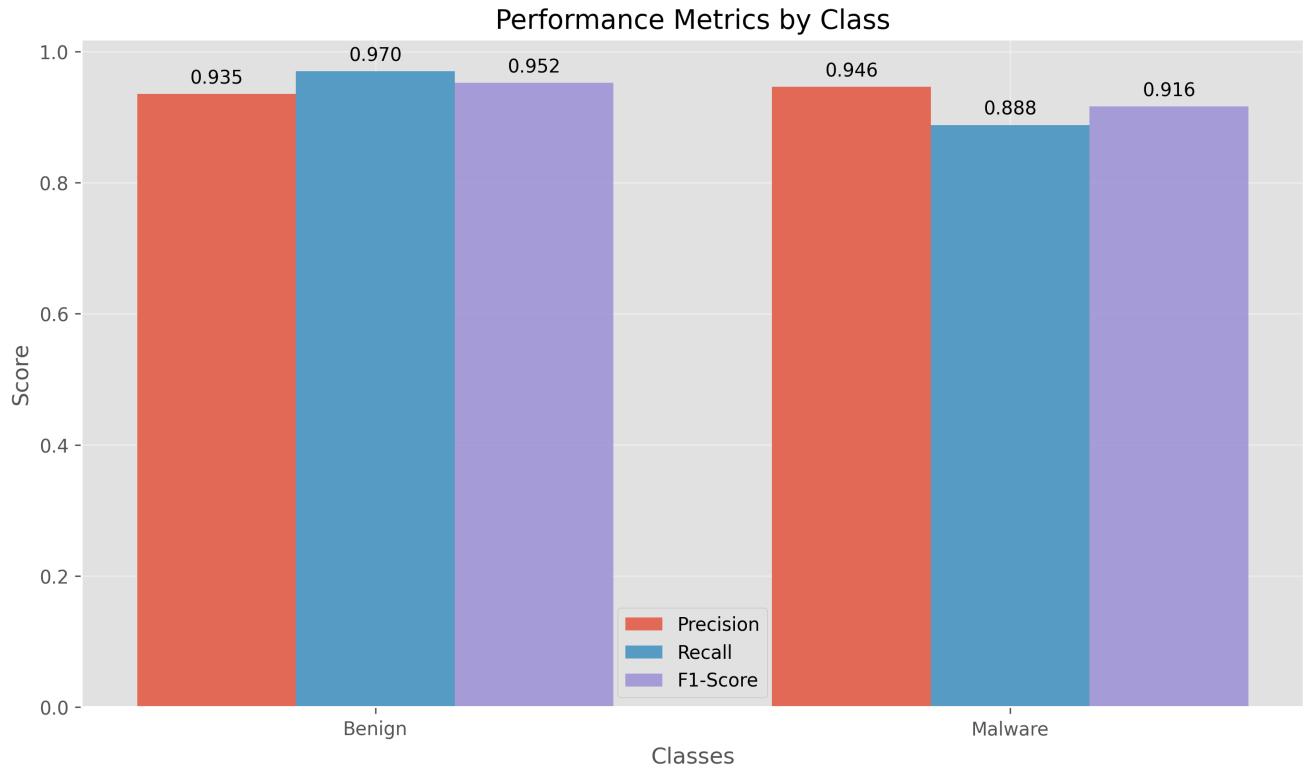
4.3 Curvas de Avaliação Precisão e Recall



Explicação:

A curva de Precisão-Recall é particularmente útil em cenários com classes desbalanceadas. Ela ilustra a relação entre a **precisão** (proporção de verdadeiros positivos entre os positivos previstos) e o **recall** (proporção de verdadeiros positivos identificados corretamente). O valor médio de precisão (Average Precision = 0.975) indica que, mesmo com alto recall, o modelo mantém uma elevada taxa de precisão, minimizando alarmes falsos.

4.4 Avaliação Por Classe



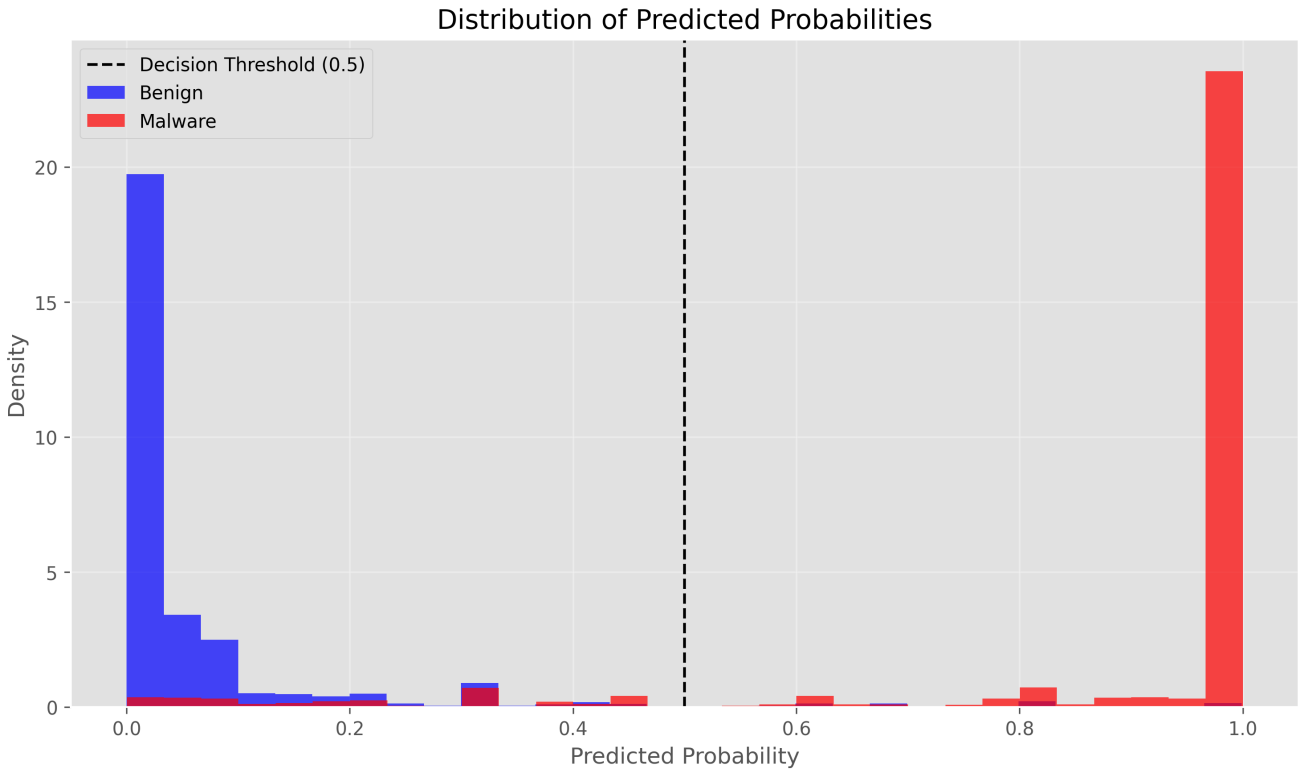
Explicação:

Este gráfico compara as métricas de avaliação — **Precisão**, **Recall** e **F1-Score** — para cada classe (Benigno e Malware), permitindo uma análise detalhada do equilíbrio do modelo entre diferentes tipos de erro.

- **Benigno**: apresenta alta **revocação (0.970)**, o que indica que quase todos os aplicativos benignos foram corretamente identificados. A **precisão (0.935)** também é elevada, significando que a maioria das previsões como benignas realmente corresponde a essa classe. O **F1-Score (0.952)** resume esse bom equilíbrio entre precisão e recall.
- **Malware**: tem uma **precisão ainda mais alta (0.946)**, o que é crucial em sistemas de segurança, pois minimiza o número de falsos positivos (benignos classificados como malware). A **revocação (0.888)**, embora ligeiramente inferior, ainda indica boa capacidade de detecção. O **F1-Score (0.916)** demonstra um desempenho sólido e consistente na identificação de ameaças.

Em conjunto, esses resultados sugerem que o modelo mantém **bom equilíbrio entre segurança (detecção de malware) e confiabilidade (baixo alarme falso)**, sendo apropriado para ambientes onde a minimização de riscos e ruído operacional é essencial.

4.4 Avaliação Por Classe



Explicação:

Este gráfico exibe a densidade das probabilidades preditas pelo modelo para cada classe (Benigno em azul, Malware em vermelho), com um **limiar de decisão fixado em 0.5** (linha tracejada).

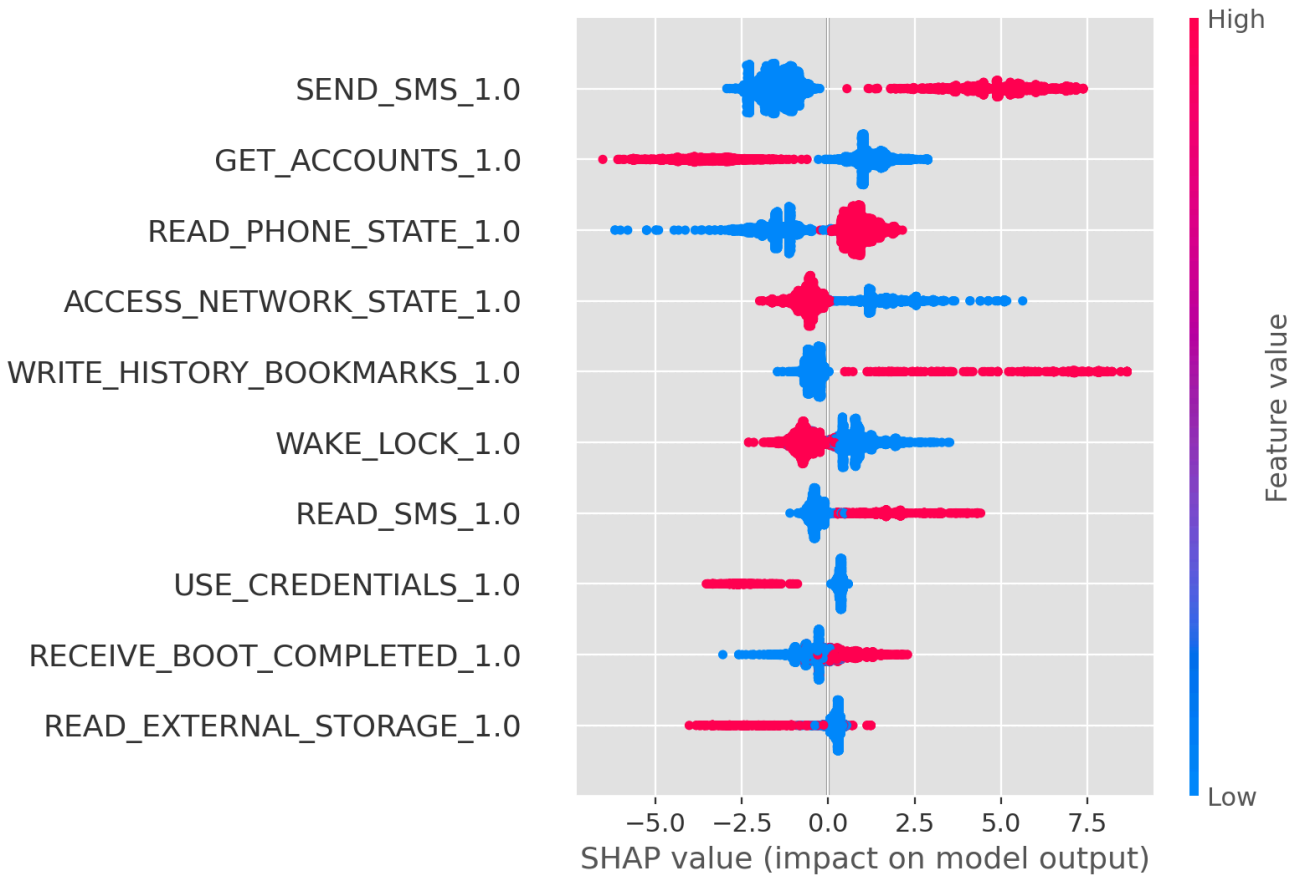
Observações importantes:

- A maioria dos exemplos **benignos** concentra-se à esquerda do limiar (probabilidades próximas de 0), indicando alta confiança do modelo ao classificá-los corretamente como não maliciosos.
- De forma análoga, a maioria dos exemplos **maliciosos** se agrupa à direita do limiar (probabilidades próximas de 1), evidenciando também alta confiança na classificação como malware.
- A separação clara entre as duas distribuições sugere **alta discriminabilidade** do modelo, ou seja, baixa ambiguidade na predição.
- A baixa sobreposição entre as curvas reduz a taxa de erros de classificação, como falsos positivos e falsos negativos.

Esse comportamento é desejável em sistemas de detecção, pois reforça que o modelo não apenas acerta as classes, mas o faz com **alta confiança estatística**, tornando a ferramenta confiável para uso em ambientes críticos.

5. 🔍 Interpretabilidade

5.1 Análise SHAP Summary Plot



Explicação:

Este gráfico resume a **influência de cada permissão Android** sobre as previsões do modelo, utilizando valores SHAP, que quantificam o impacto de cada feature na saída do classificador.

Interpretação do Gráfico:

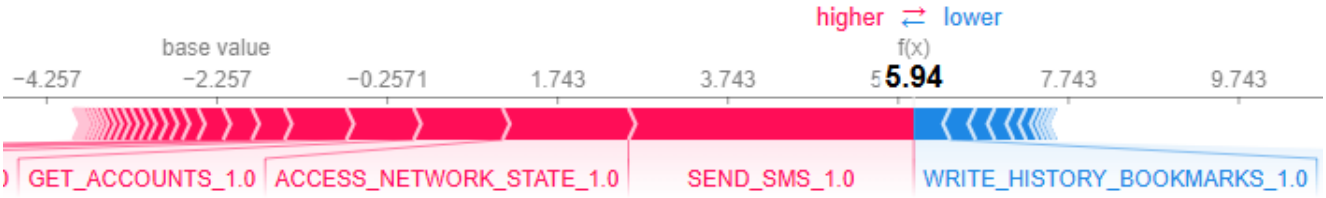
- O eixo X representa o valor SHAP, ou seja, o **impacto individual** da feature na predição. Valores positivos empurram a previsão para “malware”, enquanto valores negativos favorecem a classe “benigno”.
- Cada ponto representa uma amostra; a cor indica o valor da feature (vermelho = valor alto / presente, azul = valor baixo / ausente).
- As permissões mais relevantes incluem:
- `SEND_SMS_1.0`, `READ_PHONE_STATE_1.0`, `READ_SMS_1.0`: quando **ativas (vermelhas)**, têm forte impacto positivo na classificação como **malware**, sugerindo comportamento

malicioso.

- `GET_ACCOUNTS_1.0` e `ACCESS_NETWORK_STATE_1.0`: apresentam impacto misto, com comportamentos diferentes dependendo do contexto.
- Permissões como `RECEIVE_BOOT_COMPLETED_1.0` e `WRITE_HISTORY_BOOKMARKS_1.0` também contribuem, mas com menor intensidade.

Este gráfico permite concluir que o modelo **aprende padrões interpretáveis** e condizentes com práticas conhecidas de malware, reforçando a confiabilidade e **explicabilidade** do processo de decisão. Além disso, evidencia a importância de um subconjunto reduzido de permissões, o que pode auxiliar na **redução dimensional** e auditoria dos atributos usados.

5.2 Análise SHAP Force Plot



Explicação:

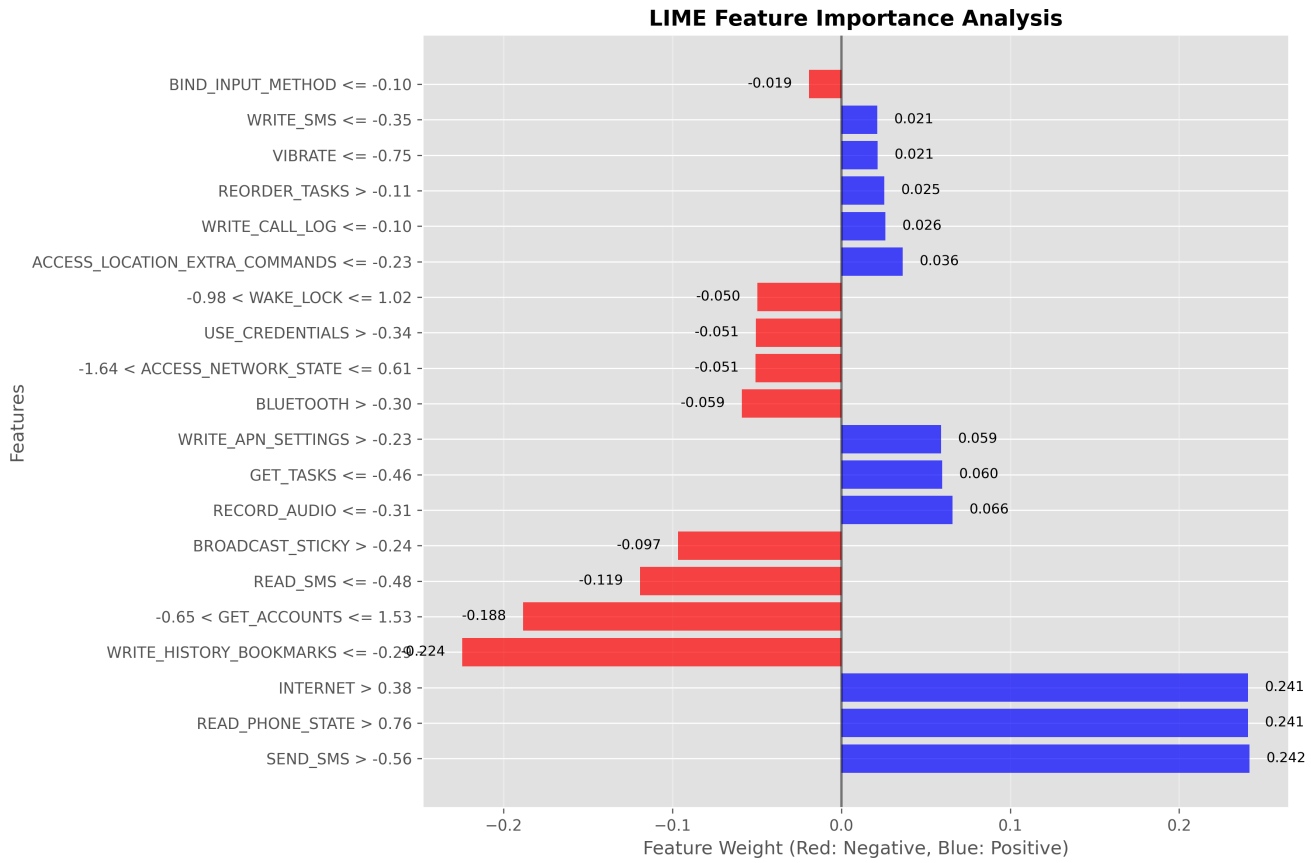
Este gráfico visualiza como os valores das features influenciaram uma predição específica do modelo. A previsão final (`f(x) = 5.94`) resulta da soma do valor base (média das predições) com os impactos acumulados de cada atributo.

Interpretação:

- **Base value**: é o valor médio da saída do modelo sem considerar nenhuma feature (referência neutra).
- **Setas vermelhas (→ higher)**: indicam features que **aumentaram** a probabilidade da amostra ser classificada como **malware**.
- `SEND_SMS_1.0`, `ACCESS_NETWORK_STATE_1.0` e `GET_ACCOUNTS_1.0` contribuíram positivamente, impulsionando a predição para a classe maliciosa.
- **Seta azul (→ lower)**: representa uma feature que **reduziu** essa probabilidade.
- `WRITE_HISTORY_BOOKMARKS_1.0` teve um impacto negativo, atuando como um fator benigno.

O valor final de **5.94** está bem acima do limiar de decisão, reforçando que o modelo classificou esta instância com **alta confiança como malware**.

5.3 Análise importância das características LIME



Explicação:

O gráfico apresenta a contribuição individual de cada feature na decisão do modelo para uma **amostra específica**, permitindo uma análise **local** da explicação. Os pesos indicam o quanto cada atributo influenciou a classificação:

- **Barras azuis (positivas)**: características que **favoreceram a predição como malware**.
- Destaques:
- `SEND_SMS > -0.56` (peso: +0.242)
- `READ_PHONE_STATE > 0.76` (peso: +0.241)
- `INTERNET > 0.38` (peso: +0.241)

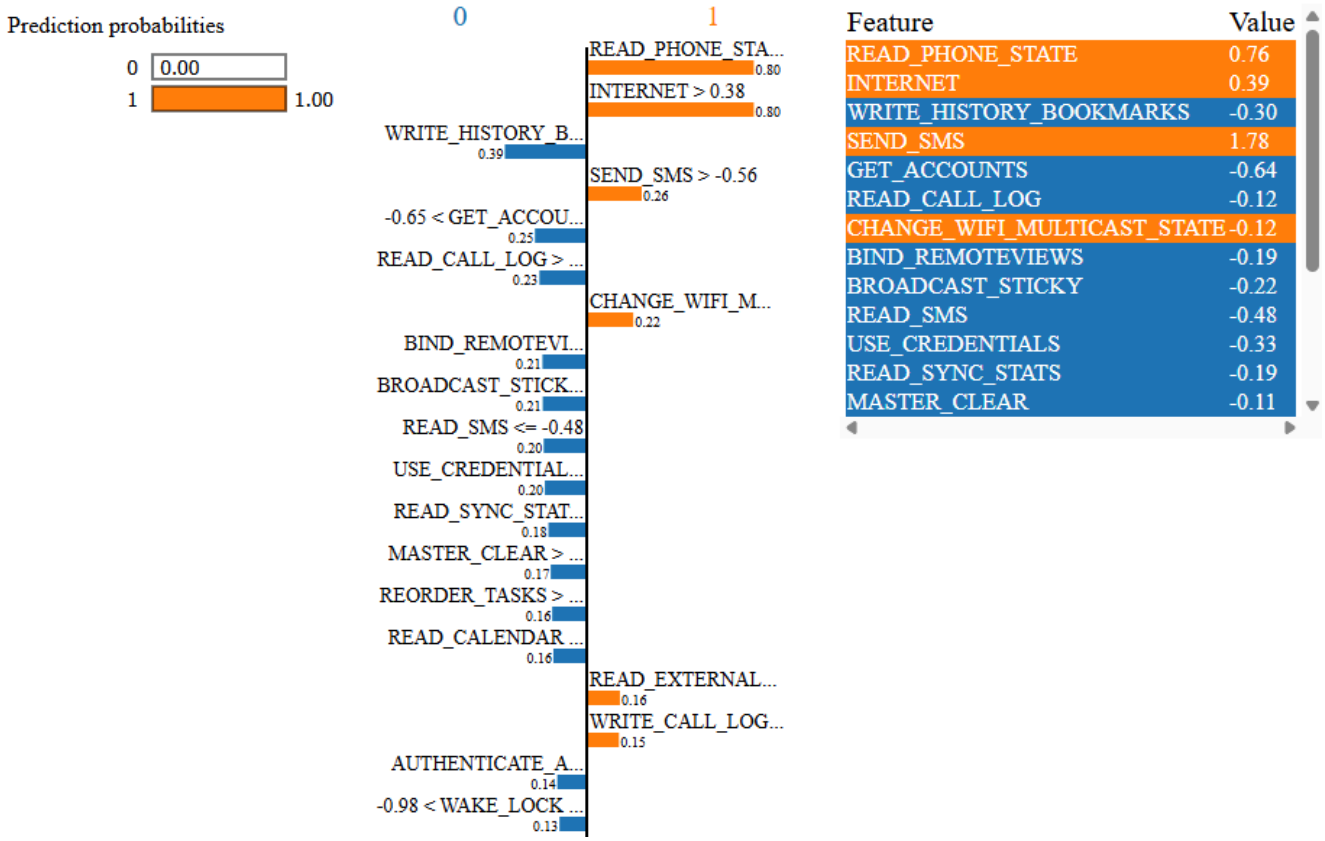
→ A presença ou valores altos dessas permissões aumentaram significativamente a probabilidade de a amostra ser classificada como maliciosa.

- **Barras vermelhas (negativas)**: características que **atuaram contra a classificação como malware**, ou seja, aproximaram a instância da classe benigna.
- Destaques:
- `WRITE_HISTORY_BOOKMARKS <= -0.22` (peso: −0.224)
- `GET_ACCOUNTS <= 1.53` (peso: −0.188)
- `READ_SMS <= -0.48` (peso: −0.119)

→ Esses atributos, ao estarem ausentes ou abaixo de determinado valor, indicaram comportamento benigno ao modelo.

Este gráfico complementa a explicação global do SHAP ao mostrar **como o modelo tomou uma decisão em um caso concreto**, oferecendo uma forma interpretável e confiável de justificar decisões individuais — fundamental em contextos como cibersegurança e auditoria.

5.4 Análise Probabilidade de Predição LIME



Insights:

O gráfico mostra como as **features ativas** (valores fornecidos à direita) impactaram a predição do modelo (barra central), contribuindo para a classificação como **malware (classe 1)** ou

benigno (classe 0).

🔶 **Principais contribuições para classe 1 (malware):**

- `READ_PHONE_STATE = 0.76`
- `INTERNET = 0.39`
- `SEND_SMS = 1.78`
- `CHANGE_WIFI_MULTICAST_STATE = -0.12`

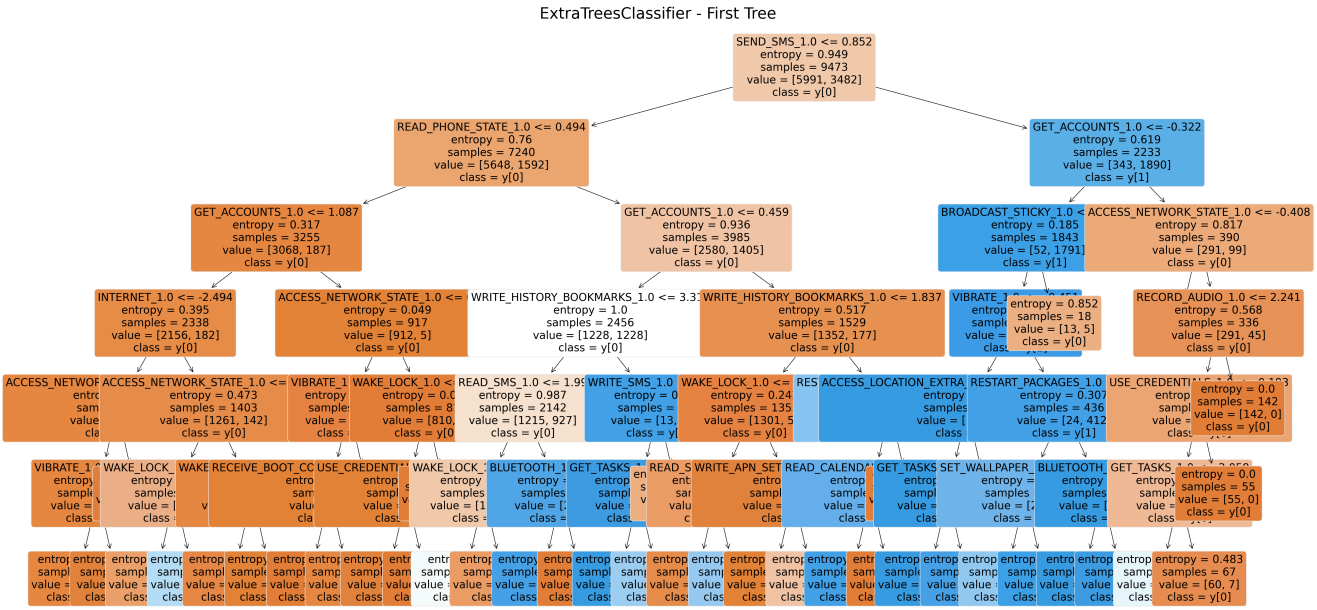
Estas permissões são **fortes indicativos de comportamento malicioso** e empurraram a predição para a classe “malware”.

🔷 **Principais contribuições para classe 0 (benigno):**

- `WRITE_HISTORY_BOOKMARKS = -0.30`
- `GET_ACCOUNTS = -0.64`
- `READ_CALL_LOG = -0.12`
- `USE_CREDENTIALS = -0.33`
- `READ_SMS = -0.48`
- `MASTER_CLEAR = -0.11`

Essas permissões, por estarem ausentes ou com valor baixo, puxaram a predição em direção à classe benigna. Mesmo assim, o modelo atribuiu **alta probabilidade (≈ 1.00)** para a classe malware, dado o maior peso das permissões maliciosas.

5.4 Análise da Árvore de Decisão



A imagem exibe a estrutura de uma única **árvore de decisão**, que é a primeira árvore de um modelo de `ExtraTreesClassifier`. Este tipo de modelo de machine learning cria uma "floresta" de múltiplas árvores e combina seus resultados, mas a visualização de uma única árvore nos permite entender a lógica de classificação que o modelo aprendeu.

A árvore funciona como um fluxograma de decisões:

- **Nós (caixas):** Cada nó interno representa uma pergunta sobre uma feature específica (uma permissão do Android).
- **Ramos (setas):** As setas indicam o caminho a seguir com base na resposta ("sim" ou "não", ou, mais precisamente, `True` ou `False` para a condição).
- **Folhas (nós finais):** Os nós na base da árvore representam a decisão final de classificação.

Dentro de cada nó, temos as seguintes informações:

- **Condição de divisão:** A regra usada para dividir os dados (ex: `SEND_SMS_1.0 <= 0.852`).
- **`entropy`:** Uma medida de impureza do nó. Um valor de 0 significa que o nó é "puro" (todas as amostras pertencem à mesma classe).
- **`samples`:** O número de aplicativos que chegaram a este nó.
- **`value`:** A distribuição das classes neste nó (ex: `[5991, 3482]` significa 5991 amostras da classe 0 e 3482 da classe 1).

- **`class`:** A classe majoritária no nó. A cor do nó (laranja para classe 0 - Benigno, azul para classe 1 - Malware) reflete essa maioria.

Principais destaques:

A análise da estrutura da árvore revela uma clara hierarquia na importância das permissões para a detecção de malware.

- **Feature Mais Importante:** A primeira decisão, no topo da árvore (nó raiz), é baseada na permissão `SEND_SMS_1.0`. Isso significa que, de todas as permissões disponíveis, esta é a que melhor consegue separar os aplicativos benignos dos maliciosos no primeiro passo.

- **Caminho Crítico para Malware:** O caminho à direita, onde `SEND_SMS_1.0` é `True` (maior que 0.852), leva a um nó que é predominantemente **Malware (azul)**, com 1890 amostras de malware contra 343 benignas. Isso indica que a solicitação da permissão para enviar SMS é um fortíssimo indicador de malícia.

- **Hierarquia de Permissões:** Para os aplicativos que *não* solicitam `SEND_SMS` (o caminho da esquerda), a próxima pergunta mais importante é sobre a permissão `READ_PHONE_STATE_1.0`. Se a resposta for sim, a suspeita aumenta. Se for não, o modelo continua a perguntar sobre outras permissões como `GET_ACCOUNTS_1.0`.

- **Pureza das Folhas:** O objetivo da árvore é terminar em folhas que sejam o mais “puras” possível (predominantemente de uma única cor). Vemos que a árvore consegue isolar grupos de malware (folhas azuis) e de aplicativos benignos (folhas laranjas) com razoável sucesso.

Interpretação:

Esta árvore de decisão torna o processo de classificação do modelo transparente e interpretável. Ela essencialmente cria um conjunto de regras “se-então” para identificar malware.

A lógica do modelo pode ser lida como um processo de triagem:

1. **Primeiro, verifique se o aplicativo envia SMS.** Se sim, a probabilidade de ser malware é muito alta.

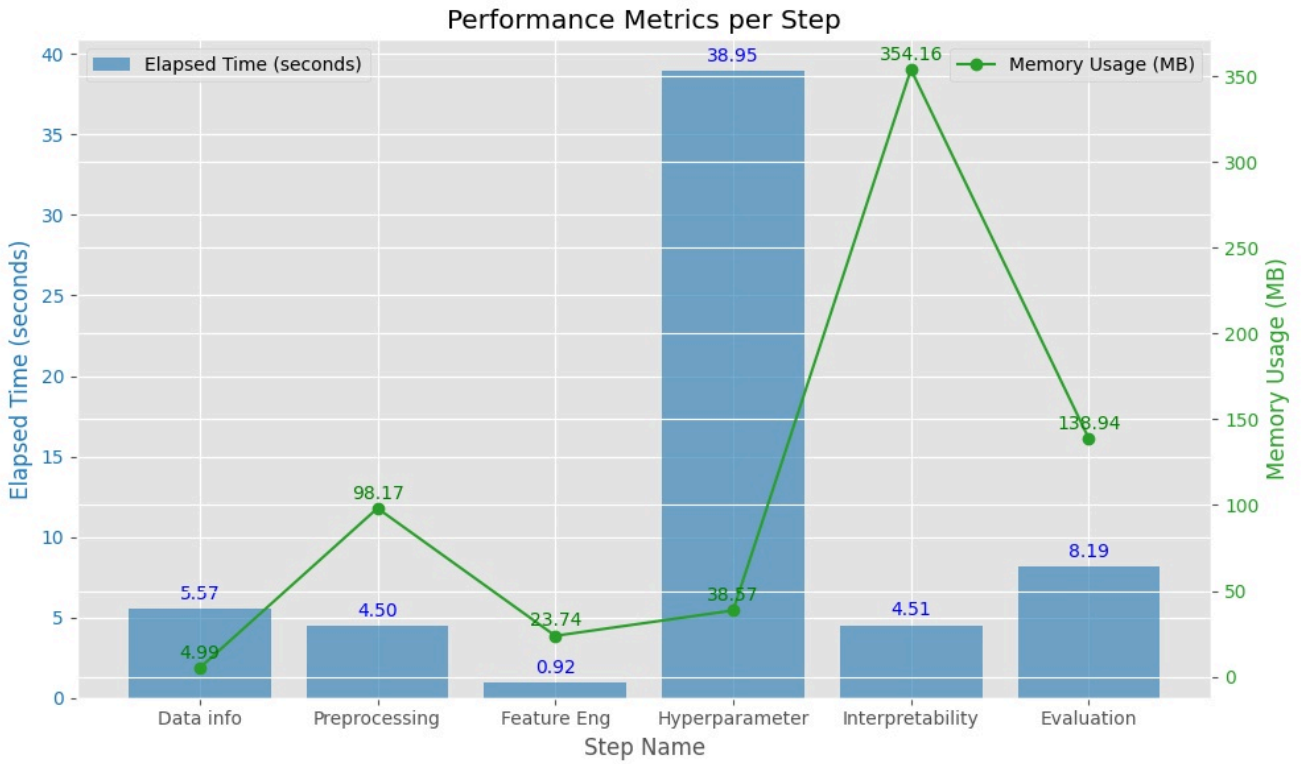
2. **Se não, verifique se ele lê o estado do telefone.** Esta é a segunda bandeira vermelha mais importante.

3. **Se não, verifique se ele acessa as contas do usuário.** E assim por diante.

As permissões que aparecem nos níveis superiores da árvore (`SEND_SMS`, `READ_PHONE_STATE`, `GET_ACCOUNTS`) são as mais impactantes para o modelo. Esta conclusão está perfeitamente alinhada com as análises de PCA anteriores, que também destacaram a importância dessas mesmas features. A árvore, no entanto, nos dá uma visão mais direta e processual, mostrando a ordem e a lógica exata das decisões que levam a uma classificação final de **Benigno** ou **Malware**.

6. Performance geral do Pipeline

5.2 Desempenho Tempo x RAM



A imagem apresentada é um gráfico de barras e linhas que detalha as métricas de performance (tempo de execução em segundos e consumo de memória RAM em MB) para cada etapa de um pipeline de Machine Learning (ML).

Análise do Gráfico:

O gráfico possui dois eixos Y:

- **Eixo Y Esquerdo (Azul):** Representa o “Elapsed Time (seconds)” (Tempo Decorrido em segundos) para cada etapa. As barras azuis mostram esses valores.
- **Eixo Y Direito (Verde):** Representa o “Memory Usage (MB)” (Uso de Memória em MB). A linha verde com marcadores circulares mostra esses valores.

As etapas do pipeline de ML estão no **Eixo X**, rotuladas como “Step Name”:

1. **Data Info**
2. **Preprocessing**
3. **Feature Engineering**
4. **Hyperparameter**
5. **Interpretability**
6. **Evaluation**

Vamos analisar cada etapa:

- **Data Info:**
  - **Tempo de Execução:** 5.57 segundos (barra azul)
  - **Uso de Memória:** 4.99 MB (marcador verde)
  - Esta etapa inicial para obter informações sobre os dados é relativamente rápida e consome pouca memória.
- **Preprocessing (Pré-processamento):**
  - **Tempo de Execução:** 4.50 segundos (barra azul)
  - **Uso de Memória:** 98.17 MB (marcador verde)
  - Embora o tempo de execução seja menor que o de “Data Info”, o consumo de memória aumenta significativamente, o que é comum em etapas de limpeza, tratamento de valores ausentes ou normalização de dados.
- **Feature Eng (Engenharia de Características):**
  - **Tempo de Execução:** 0.92 segundos (barra azul)

- **Uso de Memória:** 23.74 MB (marcador verde)
- Esta é a etapa mais rápida em termos de tempo de execução e possui um consumo de memória moderado. A criação ou transformação de características pode ser eficiente em alguns casos.

- **Hyperparameter (Otimização de Hiperparâmetros):**
  - **Tempo de Execução:** 38.95 segundos (barra azul)
  - **Uso de Memória:** 39.57 MB (marcador verde)
  - Esta é a etapa mais demorada do pipeline, consumindo quase 39 segundos. Isso é esperado, pois a otimização de hiperparâmetros (por exemplo, busca em grade, busca aleatória) envolve a treinamento e avaliação de múltiplos modelos. O consumo de memória é relativamente baixo em comparação com o pré-processamento.

- **Interpretability (Interpretabilidade):**
  - **Tempo de Execução:** 4.51 segundos (barra azul)
  - **Uso de Memória:** 354.16 MB (marcador verde)
  - Embora o tempo de execução seja razoável, esta etapa apresenta o **maior consumo de memória RAM** de todo o pipeline, atingindo mais de 354 MB. Isso pode indicar o uso de algoritmos complexos para explicar as previsões do modelo, que podem exigir a manipulação de grandes estruturas de dados.

- **Evaluation (Avaliação):**
  - **Tempo de Execução:** 8.19 segundos (barra azul)
  - **Uso de Memória:** 138.94 MB (marcador verde)
  - A etapa final, responsável por avaliar o desempenho do modelo, leva um tempo considerável e tem um consumo de memória elevado, embora menor que a etapa de interpretabilidade. Isso pode envolver o cálculo de diversas métricas e a geração de relatórios.

Pontos Chave e Insights:

- **Gargalo de Tempo:** A etapa de “Hyperparameter” é o principal gargalo em termos de tempo de execução.
- **Gargalo de Memória:** A etapa de “Interpretability” é o principal gargalo em termos de consumo de memória RAM.
- **Trade-offs:** É interessante observar que as etapas mais demoradas nem sempre são as que mais consomem memória, e vice-versa. Por exemplo, “Hyperparameter” é lenta mas não é a que mais usa memória, enquanto “Interpretability” consome muita memória em um tempo moderado.
- **Otimização:** Este tipo de gráfico é crucial para identificar áreas onde a otimização pode ser mais eficaz. Por exemplo, se a memória for um problema, focar em reduzir o consumo na etapa de “Interpretability” seria prioritário. Se o tempo for crítico, otimizar a “Hyperparameter” seria essencial.

Em resumo, a imagem fornece uma visão clara e quantitativa do desempenho de cada componente do pipeline de ML, permitindo que os desenvolvedores e engenheiros de ML identifiquem e abordem ineficiências em termos de tempo e uso de recursos.

7. 📝 Considerações Finais

O pipeline de Machine Learning “MH-AutoML” demonstra uma robustez notável na detecção de malwares em aplicações Android, com base nas permissões solicitadas. A análise detalhada dos artefatos gerados em cada etapa oferece insights valiosos sobre o comportamento do modelo e a importância das características.

Pontos Fortes do Pipeline:

- **Alta Performance:** As métricas de avaliação, como AUC (0.992 ±0.03) e F1-Score Balanceado (0.968), indicam um desempenho excepcional na classificação, minimizando tanto falsos positivos quanto falsos negativos. A matriz de confusão e as curvas ROC/Precision-Recall corroboram a capacidade discriminativa do modelo.
- **Interpretabilidade Aprofundada:** O uso de ferramentas como SHAP e LIME é crucial. O SHAP Summary Plot e o Force Plot revelam que permissões como `SEND_SMS`, `READ_PHONE_STATE` e `INTERNET` são os principais indicadores de comportamento malicioso, o que é consistente com as expectativas de segurança de aplicativos. A análise LIME complementa, fornecendo explicações localizadas para decisões

específicas, essencial para a confiança em cenários críticos como cibersegurança. A visualização da árvore de decisão de `ExtraTreesClassifier` também oferece uma interpretação clara das regras de classificação aprendidas pelo modelo.

- **Engenharia de Features Eficiente:** As etapas de seleção de características (LASSO e ANOVA) e redução de dimensionalidade (PCA) foram bem aplicadas. O biplot do PCA demonstrou uma separação clara entre as classes benignas e maliciosas, reforçando que as permissões são características discriminativas. O heatmap das componentes principais "traduz" essas componentes em padrões de comportamento de aplicativos, como "Acesso à Internet" (PC 1) e "Funcionalidade SMS" (PC 2).

- **Otimização de Hiperparâmetros Eficaz:** O processo de otimização de hiperparâmetros, utilizando Optuna, demonstrou eficiência ao alcançar um platô de desempenho elevado em poucas tentativas. A análise de importância dos hiperparâmetros ("neighbors", "nav_depth", "leaf_size", "nples_split" e "classifier") direciona o ajuste fino, e as coordenadas paralelas mostram as combinações que levam aos melhores resultados.

- **Gestão de Dados:** A análise de valores faltantes e a distribuição de classes nos conjuntos de treinamento e teste (`~1.7:1` Benigno:Malware) indicam uma preparação de dados cuidadosa, com divisão estratificada para evitar vieses.

Desafios e Oportunidades de Otimização:

- **Gargalo de Memória na Interpretabilidade:** Conforme a análise de desempenho do pipeline, a etapa de "Interpretability" consome a maior quantidade de memória RAM (354.16 MB). Embora seja fundamental para a explicabilidade, otimizações nesta fase (e.g., amostragem, técnicas mais eficientes) podem ser exploradas para reduzir o consumo de recursos, especialmente em ambientes com restrição de memória.

- **Tempo de Execução na Otimização de Hiperparâmetros:** A etapa de "Hyperparameter" é a mais demorada (38.95 segundos). Para grandes conjuntos de dados ou otimizações mais extensas, métodos como otimização bayesiana mais avançada ou a paralelização da busca podem ser considerados para acelerar o processo.

Em suma, o MH-AutoML apresenta um framework robusto e bem-sucedido para a detecção de malware, com um equilíbrio notável entre alta performance preditiva e transparência em suas decisões. As considerações sobre o consumo de recursos indicam áreas potenciais para otimização contínua, garantindo que o pipeline não apenas seja eficaz, mas também eficiente em sua execução.