Journal, 2026, 32:1, doi: 10.5753/jbcs.2026.6185
This work is licensed under a Creative Commons Attribution 4.0 International License.

Interpretable by Design: MH-AutoML for Transparent and Efficient Android Malware Detection without Compromising Performance

Joner Assolin [**Federal University of Amazonas** | ***joner.assolin@icomp.ufam.edu.br***]
Gabriel Canto [**Federal University of Amazonas** | ***gabriel.canto@icomp.ufam.edu.br***]
Diego Kreutz [**Federal University of Pampa** | ***diegokreutz@unipampa.edu.br***]
Hendrio Bragança [**Federal University of Amazonas** | ***hendrio.luis@icomp.ufam.edu.br***]
Eduardo Feitosa [**Federal University of Amazonas** | ***efeitosa@icomp.ufam.edu.br***]
Angelo Nogueira [**Federal University of Pampa** | ***angelonogueira.aluno@unipampa.edu.br***]
Vanderson Rocha [**Federal University of Amazonas** | ***vanderson@ufam.edu.br***]

Institute of Computing, Federal University of Amazonas (UFAM), Av. Gen. Rodrigo Otávio, 6200, Coroado I, Manaus, AM, 69067-005, Brazil.

Received: 29 June 2025 • **Accepted:** 25 March 2026 • **Published:** 28 August 2026

Abstract. Malware detection in Android systems requires both cybersecurity domain knowledge and expertise in machine learning (ML). Automated Machine Learning (AutoML) aims to simplify ML development by automating key stages of the pipeline, including pipeline construction, algorithm selection, and hyperparameter optimisation, thereby reducing the need for specialised expertise. However, most general-purpose AutoML frameworks operate largely as black-box systems with limited transparency, interpretability, and experiment traceability. These limitations are particularly problematic in security-sensitive applications, where understanding model behaviour and ensuring reproducibility are essential. To address these challenges, we present MH-AutoML, a meta-heuristic-based AutoML framework designed specifically for Android malware detection. MH-AutoML automates the complete ML pipeline, covering data preprocessing, feature engineering, model selection, and hyperparameter optimisation. In addition, the framework incorporates built-in capabilities for interpretability, debugging, and experiment tracking, which are often absent or only partially supported in existing AutoML solutions. We perform a large-scale empirical evaluation comparing MH-AutoML with seven established AutoML frameworks (Auto-sklearn, AutoGluon, TPOT, HyperGBM, AutoPyTorch, LightAutoML, and MLJAR). The comparison is conducted across nine Android malware benchmark datasets evaluated under both original and class-balanced distributions using 10 random seeds, resulting in a total of 1,440 experimental runs. Using the Matthews Correlation Coefficient (MCC) as the primary evaluation metric, together with Friedman–Nemenyi statistical tests and pairwise Win/Tie/Loss analysis, the results reveal several key findings. First, MLJAR and LightAutoML form a consistently top-performing tier, with MLJAR achieving the highest aggregate dominance score across the analyses. Second, LightAutoML offers the most favourable efficiency–performance trade-off, reaching comparable predictive quality in approximately 9–17 minutes compared with MLJAR's execution time of about one hour. Third, MH-AutoML achieves competitive recall performance, ranking first on three datasets including Adroit, Androcrawl, and MH-100, although this performance is accompanied by reduced MCC values due to elevated false-positive rates. Fourth, class balancing substantially increases the statistical separation among frameworks, doubling the Friedman test statistic and revealing fragile behaviour in certain tools. AutoPyTorch is the most notable case, collapsing from third position in the Original setting to the last rank under balanced distributions. Finally, no single framework dominates across all datasets, which reinforces the importance of selecting AutoML frameworks according to the specific characteristics of each dataset.

Keywords: Automated Machine Learning, AutoML, Android Malware Detection, Cybersecurity, Machine Learning Pipeline, Explainable Artificial Intelligence (XAI), Transparency in AI, Interpretability, Hyperparameter Optimization, Feature Engineering, Comparative Evaluation, MLOps, Model Selection, Domain-Specific AutoML, Black-Box Systems.

1 Introduction

The increasing sophistication of Android malware has led to a growing reliance on machine learning (ML) for detection, as traditional signature-based methods often struggle to keep pace with evolving threats. The Android ecosystem, due to its global adoption and open application model, has become a primary target for malicious actors. Contemporary malware families employ techniques such as code obfuscation, dynamic loading, reflection, and permission abuse, making static detection mechanisms increasingly insufficient.

While ML offers promising capabilities, developing effective detection models requires substantial expertise in statistics, programming, and cybersecurity. This creates a significant barrier for practical adoption, particularly in environments where such specialized knowledge is scarce. In security-sensitive domains, this barrier is even more critical, as incorrect model behavior may lead to undetected threats or excessive false positives that impact operational workflows.

Automated machine learning (AutoML) [Truong *et al.*,

2019a] emerges as a potential solution by streamlining the ML workflow. By automating critical stages of the pipeline (i.e., data preprocessing, feature engineering, model selection, and hyperparameter optimization) AutoML reduces the technical burden, allowing practitioners to focus on domain-specific challenges rather than implementation details Truong *et al*. [2019a]; Wu *et al*. [2024]; Zhang *et al*. [2025]. However, despite the rapid expansion of AutoML tools, their effectiveness and applicability in specialized domains like malware detection remain uncertain.

One key challenge lies in the diversity of AutoML solutions, which range from general-purpose frameworks to domain-specific implementations, making direct comparisons difficult [Zöller and Huber, 2021]. Additionally, critical aspects of the ML pipeline, such as explainable artificial intelligence (XAI), often receive insufficient attention [Carvalho *et al*., 2019; Vilone and Longo, 2020; Barbudo *et al*., 2023; Salehin *et al*., 2024]. Many existing AutoML tools also exhibit limitations in transparency, debugging support, transfer learning, scalability, and pipeline customization [Santu *et al*., 2021; Barbudo *et al*., 2023; Baratchi *et al*., 2024; Tian and Che, 2024; Salehin *et al*., 2024]. These shortcomings are particularly problematic in security-sensitive applications like malware detection, where interpretability and operational trustworthiness are essential.

Although several benchmarking studies have evaluated AutoML frameworks across different domains, few focus specifically on Android malware detection, and even fewer incorporate structured evaluation of transparency and interpretability alongside predictive performance. Most prior comparisons emphasize metrics such as accuracy or AUC, without systematically examining class imbalance behavior, experiment reproducibility, or internal pipeline transparency.

To bridge these gaps, we introduce MH-AutoML[1], a domain-specific AutoML framework tailored for Android malware detection. Unlike general-purpose solutions, MH-AutoML provides a fully automated pipeline that integrates data cleaning, feature engineering, model selection, and hyperparameter tuning, while also emphasizing transparency, interpretability, and debugging capabilities. These features, often overlooked in existing tools, are critical for ensuring that security analysts can validate and trust detection outcomes.

The main contributions of this work include:

1. The design and implementation of MH-AutoML, an AutoML framework specifically optimized for Android malware detection. The framework supports the entire ML pipeline, including data preprocessing, feature engineering, algorithm selection, and hyperparameter tuning. MH-AutoML was developed with a focus on real-world applicability in cybersecurity, offering mechanisms that facilitate transparency and model validation;
2. Novel support for transparency, interpretability, experiment tracking, and pipeline customization. To assess these aspects, we introduce a custom evaluation model based on scores derived from questions structured around five key dimensions: Functional Description, Statistical Analysis, Algorithmic Transparency, Interpretability, and Internal Analysis. This evaluation reveals MH-AutoML's superiority over existing AutoML tools in terms of explainability and user insight;
3. A comprehensive empirical evaluation comparing MH-AutoML against seven widely used general-purpose AutoML frameworks, including Auto-Sklearn[2], AutoGluon[3], TPOT[4], HyperGBM[5], Auto-PyTorch[6], LightAutoML[7], and MLJAR[8]. This evaluation uses both imbalanced and balanced datasets with unique samples to better assess the robustness of each tool under class imbalance conditions. We report comparative analyses on key metrics, such as recall, Matthews Correlation Coefficient (MCC), and execution time, across original, balanced, and imbalanced datasets. The results reinforce MH-AutoML's consistent performance and resilience in diverse and challenging data scenarios.

Our results indicate that MH-AutoML delivers consistently high recall with competitive computational efficiency, making it a practical and explainable option for real-world cybersecurity applications.

Unlike prior work, MH-AutoML is the first domain-specific AutoML framework for Android malware detection that simultaneously integrates interpretability, transparency scoring, system monitoring, and statistical validation under imbalanced and balanced settings.

2 AutoML Pipeline

The ML pipeline follows a structured sequence of steps that transform raw data into predictive models. Traditional approaches require manual execution of data preprocessing, feature selection, model choice, hyperparameter tuning, and evaluation, demanding significant technical expertise and iterative experimentation. AutoML automates these steps through algorithmic exploration of different technique combinations, as illustrated in Figure 1 [Truong *et al*., 2019a].

AutoML pipelines reduce human intervention by systematically searching for optimal configurations using methods like Bayesian optimization, genetic programming, and reinforcement learning [Hutter *et al*., 2019; Doke and Gaikwad, 2021; Wu *et al*., 2024; Zhang *et al*., 2025]. This automation makes ML more accessible to non-experts while maintaining rigorous optimization standards. Figure 2 details these automated steps and their implementation in typical AutoML systems.

[1] This work significantly extends our previous contribution Assolin *et al*. [2024], recognized as an outstanding artifact at SBSeg'24, by incorporating recent literature, evaluating performance on balanced datasets to address class imbalance, and introducing a novel interpretability assessment framework. Key enhancements include an expanded comparative analysis of AutoML tools using robust metrics (recall, MCC, execution time), a custom transparency evaluation model across five dimensions, and in-depth validation of *MH-AutoML*'s effectiveness in imbalanced scenarios, demonstrating its consistent performance and superior explainability compared to existing solutions.

[2] https://automl.github.io/auto-sklearn
[3] https://auto.gluon.ai
[4] http://epistasislab.github.io/tpot
[5] https://github.com/DataCanvasIO/HyperGBM
[6] https://automl.github.io/Auto-PyTorch
[7] https://lightautoml.readthedocs.io
[8] https://supervised.mljar.com

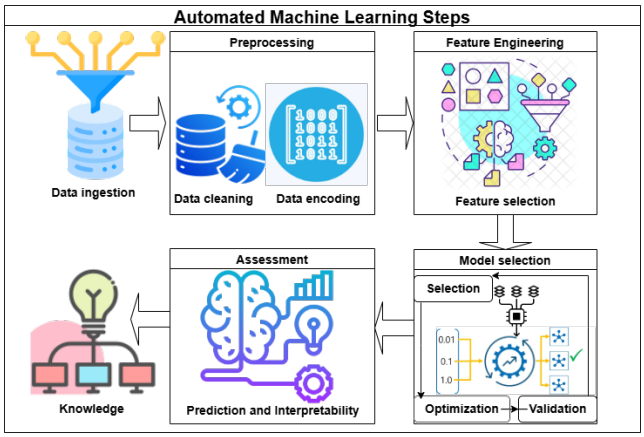

Figure 1. AutoML execution steps.

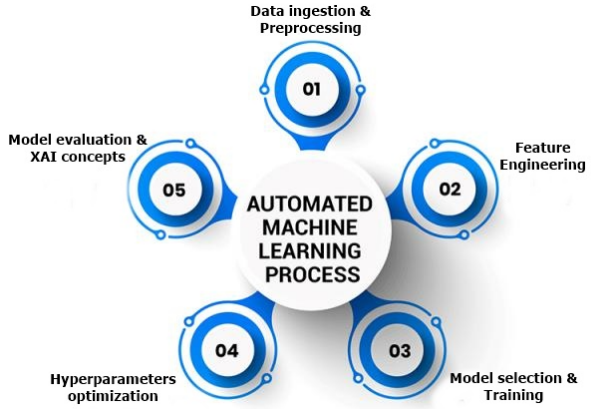

Figure 2. AutoML process.

Data preprocessing forms the foundation of the pipeline, where raw data undergoes transformations to improve quality and consistency. Common preprocessing tasks include:

- Imputation of missing values using statistical estimates;
- Treatment of outliers through removal or dampening techniques;
- Data transformations including scaling, normalization, and categorical encoding.

Automated systems select these preprocessing techniques through heuristic or optimization-driven approaches, reducing bias risks while maintaining model performance [Giovanelli *et al.*, 2022; Bilal *et al.*, 2022].

Feature engineering builds upon preprocessed data by creating or modifying attributes to enhance predictive power. Typical operations include transforming existing features through encoding, discretization, or dimensionality reduction, and generating new features by combining attributes or extracting statistical properties. AutoML implementations typically incorporate domain-specific knowledge, particularly in specialized applications like Android malware detection where features like permission patterns and obfuscation indicators prove valuable.

Model selection involves evaluating different algorithms against the problem requirements and dataset characteristics. AutoML systems automate this comparison across various model classes including decision trees, ensemble methods, neural networks, and linear models, using cross-validation to assess performance objectively.

The subsequent hyperparameter optimization phase systematically explores configuration spaces using methods like grid and random search for comprehensive exploration, and bayesian optimization and genetic algorithms for efficient search strategies. These methods automatically identify parameter sets that maximize target metrics such as accuracy or F1-score [Wu *et al.*, 2019; Yang and Shami, 2020].

Interpretability features allow understanding and validating model decisions through techniques like LIME [Ribeiro *et al.*, 2016] and SHAP [Lundberg and Lee, 2017]. In AutoML implementations, these explainability components help monitor pipeline construction and facilitate operational validation.

The final prediction analysis stage provides comprehensive model evaluation through:

- Performance metrics including accuracy, precision, recall, and AUC;
- Visualizations like confusion matrices and ROC curves;
- Comparative analysis of alternative approaches.

Automation in this phase enables continuous model improvement and maintains experiment histories as new data becomes available.

3 Related Work

We organize our literature review to systematically explore the AutoML landscape through four interconnected sections. In Section 3.1, we examine comparative studies that evaluate existing AutoML tools across various domains and performance metrics, establishing benchmarks for assessing tool capabilities. Building on this foundation, Section 3.2 documents novel approaches that advance current AutoML methodologies, while also cataloging publicly available tools to provide practical references for practitioners and researchers.

Section 3.3 focuses on interpretability techniques and their implementation in AutoML systems, addressing the growing need for explainability. This organizational structure guides readers through a logical progression from performance evaluation to methodological innovations, followed by interpretability considerations and practical tool overviews.

Our approach helps readers first understand evaluation standards, then explore emerging developments, examine explainability solutions, and finally discover available implementations. Together, these sections provide a comprehensive perspective on the current AutoML ecosystem while identifying key research directions and practical applications.

3.1 Benchmarking

We organize our analysis of AutoML benchmarking studies in Table 1, which compares existing tools across various domains and performance metrics. These studies naturally divide into two groups based on their data sources: those using real-world datasets and those employing synthetic data.

The real-world data studies span multiple application domains, ranging from general machine learning tasks to specialized areas such as civil engineering Oliveira *et al.* [2024] and image processing Hariri-Ardebili *et al.* [2024]. These investigations show substantial variation in their experimental

Table 1. Benchmarking studies on AutoML tools and frameworks.

Ref.	Datasets	Domain	Type	Nº Tools	Compared Tools	Metrics	Validation
Truong *et al*. [2019b]	300	Non-specific	Real data	7	Ludwig, H2O-Automl, TPOT, Darwin, Auto-sklearn, Auto-Keras, Auto-ml	Accuracy, MSE, F1 score	Holdout
Kundu *et al*. [2021]	2	Non-specific	Real data	2	AutoGluon-Tabular, Microsoft NNI	AUC, ROC curve	Holdout
Ferreira *et al*. [2021]	12	Non-specific	Real data	8	Auto-Keras, Auto-PyTorch, Auto-Sklearn, Auto-Gluon, H2O AutoML, rminer AutoML, TPOT, TransmogrifA	MAE, AUC, F1 score	10-fold Cross validation
Gijsbers *et al*. [2024]	104	Non-specific	Real data	10	AutoGluon, Auto-Sklearn 1/2, FLAML, GAMA, H2O AutoML, LightAutoML, MLJAR, NaiveAutoML, TPOT	AUC, log loss, RMSE, time	Cross validation
Naser [2023]	6	Non-specific	Real data	5	BigML, DataRobot, Dataiku, Exploratory, RapidMiner	Accuracy, AUC, Logloss, ROC, RMSE, MAE, R2	10-fold Cross validation
Wever *et al*. [2021]	24	Non-specific	Real data	5	SMAC, Hyperband, BOHB, GGP, HTN-BF	Execution time, F1 score	10-fold Cross validation
Oliveira *et al*. [2024]	3	Image processing	Real data	3	Google Vertex AI, NVIDIA TAO, AutoGluon	AP, mAP	Cross validation
Hariri-Ardebili *et al*. [2024]	9	Civil Engineering	Real data	1	–	MAE, MSE, RMSE, R2, RMSLE, MAPE	10-fold Cross validation
Sırt and Eyüpoğlu [2025]	3	Non-specific	Synthetic data	3	Manual FE, Standard AutoML, Transfer Learning AutoML	Accuracy, F1-score, Sensitivity, Specificity, Time, Memory	Cross validation
da Silva *et al*. [2024]	100	Non-specific	Synthetic data	4	AutoML4Clust, Autocluster, cSmartML, ML2DAC	Adjusted rand index, Silhouette, Time, Memory	Cross validation
Trirat *et al*. [2024]	14	Non-specific	Synthetic data	5	Human Models, AutoGluon, GPT-3.5, GPT-4, DS-Agent	Composite score, Norm. perf.	Cross validation

design, particularly in the number of datasets used. Some studies employ extensive collections, with Truong *et al*. [2019b] evaluating 300 datasets, Gijsbers *et al*. [2024] analyzing 104, and Ferreira *et al*. [2021] working with 12. These comprehensive evaluations typically compare 7 to 10 different tools using standard performance metrics like AUC, F1-score, and RMSE. Other researchers like Kundu *et al*. [2021], Naser [2023], and Wever *et al*. [2021] take more focused approaches with fewer datasets but similar evaluation frameworks. In domain-specific applications, we observe the use of specialized metrics such as mAP, R^2, and the Silhouette Coefficient to better assess performance in particular contexts.

Studies using synthetic data [Sırt and Eyüpoğlu, 2025; Trirat *et al*., 2024; da Silva *et al*., 2024] tend to work with smaller datasets (typically 3 to 100 samples) but explore more specialized aspects of AutoML systems. These investigations examine capabilities like feature engineering, transfer learning, and clustering performance, while expanding their evaluation criteria to include computational efficiency measures such as runtime and memory usage. Tools like AutoML4Clust and GPT-based models feature prominently in these synthetic data studies. Across both categories, we observe the frequent use of platforms such as AutoGluon and Auto-Sklearn, along with a clear progression in validation methods from basic holdout approaches to more sophisticated cross-validation techniques.

Our analysis reveals distinct benchmarking approaches tailored to different research goals. Real-world studies emphasize practical applicability across diverse domains, while synthetic data investigations enable deeper analysis of specific system capabilities and computational characteristics. The recurring presence of certain tools across multiple studies indicates their established position in the AutoML landscape, and the evolving validation methodologies demonstrate increasing rigor in evaluation practices. These patterns provide valuable insights into current trends and future directions for AutoML benchmarking research.

3.2 AutoML tools

In Table 2, we systematically organize AutoML tools, distinguishing between those that include comparative evaluations and those introducing new solutions without benchmarking against existing approaches. This categorization helps reveal methodological differences between general-purpose and domain-specific AutoML solutions.

Our analysis shows that general-purpose tools typically employ comparative evaluations against established frameworks. Works like Zimmer *et al*. [2020]; LeDell and Poirier [2020]; Erickson *et al*. [2020] use standardized validation methods including 10-fold cross-validation, with Molino *et al*. [2019] as the notable exception in its evaluation methodology. While these tools demonstrate robust performance benchmarking, most provide limited interpretability features. Only LeDell and Poirier [2020] and Erickson *et al*. [2020] incorporate explanation methods, with LeDell and Poirier [2020] uniquely combining TreeSHAP and Individual Conditional Expectation plots with MLflow-based version control to enhance transparency and reproducibility.

Domain-specific solutions for applications in medicine [Nasimian, A. et. al., 2024], geology [Ye *et al*., 2025], and spectroscopy [Neto *et al*., 2020] exhibit distinct characteristics. These tools generally focus less on comparative evaluations but emphasize domain-appropriate interpretability features. Similar to the general-purpose category, LeDell and Poirier [2020] remains the only solution implementing both traceability and version control capabilities in this group.

We observe an apparent trade-off in current AutoML de-

Table 2. Survey of AutoML implementations across application domains: tools, tasks, interpretability, metrics, and validation

Ref.	App. & Datasets	Compared Tools	Tasks	Interpretability Method	Trace.	Transp.	Debug	Metrics	M. Sys	Validation Method
Zimmer *et al.* [2020]	General (35)	Auto-Keras, Auto-Sklearn, Hyperopt-sklearn, Auto-Net2.0, AutoGluon	Classification, Regression, Forecasting	–	–	T1, T2	D1,D2	Avg. validation accuracy, Mean relative regret	No	Holdout
Erickson *et al.* [2020]	General (50)	AutoGluon	Classification, Regression	Feature importance	–	T1, T2	D1,D2	RMSLE, R^2, MAE, Log-loss, AUC, Gini	Exec. Time, Memory	10-fold CV
LeDell and Poirier [2020]	General (44)	XGBoost GBM, H2O RF, H2O XRT, H2O GBM, H2O DL, H2O GLM	Classification, Regression	Feature importance, PD, TreeSHAP, ICE	H2O Flow, MLflow	T1, T2	D1,D2	AUC, Logloss, MPCE, RMSE	Exec. Time, Memory, Disk, Network	10-fold CV
Molino *et al.* [2019]	General (–)	TensorFlow, Keras, PyTorch, Ludwig	Classification, Regression	–	MLflow, Comet, WandB	T1, T2	D1,D2	–	Exec. Time, Memory, Disk, Network	–
Neto *et al.* [2020]	Infrared (3)	Auto-Keras	Classification, Regression, Clustering	Feature importance, IRT	–	T1, T2	D1	Accuracy	No	Holdout
Nasimian, A. et. al. [2024]	Medical (6)	–	Classification, Regression, Clustering, Forecasting	Feature and Permutation importance, SHAP, LIME	–	T1, T2	D1	HL, KMCE, CS, Acc, AUC, CK, F1 score, Jacc, MCC, NLR, NPV, Prec, Sens, Spec	No	5-fold CV
Ye *et al.* [2025]	Geological (1)	–		SHAP, Feature importance	–	–	–	EUR, CV score	No	5-fold CV
Kovalevsky *et al.* [2024]	Fitness (1)	–	Classification	–	–	–	–	Accuracy, Precision, Recall, AUC, F-score	No	Holdout
Singh *et al.* [2024]	Groundwater (7)	–	Classification	–	–	–	–	R, RMSE, BIAS	No	CV
This work	Android Malware (9)	MlJar, AutoSklearn, AutoGluon, TPOT, LightAutoML, AutoPyTorch, HyperGBM	Classification	SHAP, LIME, Feature importance	MLflow	T1, T2	D1,D2	Recall, Accuracy, F1 Score, Precision, MCC, ROC AUC	Exec. Time, Memory, Disk, Network	Holdout + 5-fold CV

velopment between rigorous performance benchmarking and comprehensive interpretability features. General-purpose tools tend to prioritize establishing performance standards through systematic comparisons, while domain-specific solutions concentrate more on explanation capabilities tailored to their application contexts.

Examining operational requirements across all evaluated tools, we identify several limitations in meeting basic transparency and debugging standards. Only five tools satisfy fundamental transparency criteria including documentation of algorithm relationships (T1) and hyperparameter configurations (T2). Fewer provide complete pipeline logs (D1) and specialized error reports (D2) for debugging purposes. The subset offering extended system monitoring (RAM, disk I/O, network metrics) shrinks to just three implementations: [Neto *et al*., 2020; Erickson *et al*., 2020; LeDell and Poirier, 2020], indicating a significant gap in production-ready features.

These findings suggest that while the AutoML field is making progress in both performance benchmarking and domain-specific interpretability, comprehensive solutions addressing both aspects remain scarce. The limited availability of operational features like monitoring and debugging in most tools points to an important area for future development to enable effective real-world deployment.

In Table 3, we present an overview of publicly available AutoML tools, organized in six criteria. We identify each tool by name, specify its license type (open-source or proprietary), note the primary programming language used, indicate CLI availability, describe its application scope (general-purpose or domain-specific), and list the developing organization.

Our analysis shows Python as the primary language for 12 of the 15 tools listed, aligning with its established position in data science and machine learning communities. Most tools adopt open-source models with modular architectures supporting customization and extension. Tools like Auto-sklearn, AutoGluon, and TPOT demonstrate this flexibility, allowing users to modify components from data preprocessing to model optimization.

We note that 10 of the 15 tools provide native CLI support, simplifying their integration into automated workflows. While most solutions target general-purpose applications, we identify specialized tools including Darwin and NVIDIA TAO for computer vision tasks, and AutoML4Clust for clustering analysis. The development ecosystem includes major technology companies (e.g., AWS, Google, Microsoft, NVIDIA), startups (e.g., V7 Labs), and academic institutions (e.g., Uni Freiburg, UPenn, DATA Lab).

This variety reflects both the maturity of the field and growing interest from various sectors. However, we note a gap: while most tools adopt general-purpose approaches, few specialize in particular domains. Darwin and NVIDIA TAO focus on computer vision, while AutoML4Clust addresses clustering tasks.

The predominance of generalist tools suggests opportunities for more specialized solutions. In this context, MH-AutoML emerges as a promising development for Android malware detection, incorporating domain-specific knowledge from cybersecurity and Android app analysis to improve feature selection, modeling, and result interpretation.

Again, we observe growing research on domain-specific AutoML tools [Oakes *et al*., 2024; Singh *et al*., 2024; Krzywanski *et al*., 2024; Schaller *et al*., 2025; Zheng *et al*., 2023; Barbudo *et al*., 2023; Salehin *et al*., 2024; Baratchi *et al*., 2024]. For example, AutoML-GWL [Singh *et al*., 2024] addresses groundwater level prediction challenges by automatically selecting and optimizing models while incorporating hydrogeological factors. The framework maintains interpretability through SHAP analysis and demonstrates strong performance across diverse aquifer systems, proving valuable for water resource management and drought prediction applications.

3.3 Interpretability in AutoML

We present in Table 4 a comprehensive overview of interpretability methods and tools, along with relevant survey literature. Our analysis reveals three main categories of contributions in this field. Foundational works like LIME and SHAP established core theoretical frameworks for model explanations. Subsequent studies such as [Zöller *et al*., 2023; Bifarin and Fernández, 2024; Garouani and Bouneffa, 2023] developed integrated approaches combining multiple techniques including SHAP, LIME, partial dependence plots, and feature importance. Comprehensive surveys including [Yuan *et al*., 2024; Tian and Che, 2024; Karmaker S. et. al., 2021] provide systematic reviews of current advancements while identifying research gaps.

General-purpose interpretability tools have evolved significantly since the introduction of LIME [Ribeiro *et al*., 2016] and SHAP [Lundberg and Lee, 2017]. Recent work in this area focuses on combining existing techniques into novel methodologies [Amirian *et al*., 2021], developing visualization tools [Zöller *et al*., 2023; Narkar *et al*., 2021], and creating unified frameworks [Garouani and Bouneffa, 2023]. Domain-specific applications show particular innovation, with tailored solutions emerging for fields like image processing [Moayeri *et al*., 2024] and healthcare [Bifarin and Fernández, 2024].

The survey literature reveals strong interest in healthcare applications, with multiple studies examining interpretability techniques in this domain [Hasan *et al*., 2024; Mahmood *et al*., 2025; Tian and Che, 2024; Yuan *et al*., 2024]. Broader surveys like Karmaker S. et. al. [2021] systematically review AutoML progress, while Amirian *et al*. [2021] synthesize recent innovations across applications. Our technical assessment shows that 62% of studies use basic pre-model interpretability methods compared to 38% employing advanced post-hoc approaches. Interactive explanation features appear in only 23% of solutions despite their demonstrated value for model debugging.

This study extends previous work through systematic evaluation of seven major AutoML tools (MLJar, AutoSklearn, AutoGluon, TPOT, LightAutoML, AutoPyTorch, and HyperGBM) across nine Android-specific datasets. Our research represents the most comprehensive domain-specific comparison to date in terms of both tool coverage and dataset specialization. Additionally, we implement a unique combination of interpretability methods with transparency mechanisms, debugging support, performance tracking, and system monitoring metrics - features rarely found together in existing solutions but critical for production environments.

Table 3. Overview of publicly available AutoML tools: license, programming language, CLI, and scope.

Tool	License	Language	Native CLI?	Scope	Company/Org
Auto-Keras	MIT	Python	Yes	General purpose	DATA Lab
Auto-Sklearn	BSD-3	Python	Yes	General purpose	Uni Freiburg
AutoGluon	Apache 2.0	Python	Yes	General purpose	AWS
H2O AutoML	Apache 2.0	Java	Yes	General purpose	H2O.ai
LightAutoML	Apache 2.0	Python	Yes	General purpose	Sberbank
TPOT	LGPL	Python	Yes	General purpose	UPenn
Ludwig	Apache 2.0	Python	Yes	General purpose	Uber
AutoML4Clust	MIT	Python	No	Clustering analyses	Academia
FLAML	MIT	.NET	No	General purpose	Microsoft
TransmogrifAI	BSD-3	Scala	No	General purpose (structured data)	Salesforce
Darwin	Proprietary	Python	No	Computer Vision	V7 Labs
DataRobot	Proprietary	R	No	General purpose	DataRobot Inc.
MLJAR	Proprietary*	Python	Yes	General purpose	MLJAR
Vertex AI	Proprietary	Python	Yes	General purpose	Google
NVIDIA TAO	Proprietary	Python	Yes	Computer Vision	NVIDIA

Table 4. Overview of interpretability methods in AutoML-related studies.

Ref.	Domain	Interpretability method	Brief description	Type
Ribeiro *et al.* [2016]	Non-specific	Local Interpretable Model-agnostic Explanations (LIME)	A method for explaining models by utilizing representative individual predictions and their explanations in a non-redundant manner.	proposed
Lundberg and Lee [2017]	Non-specific	SHapley Additive exPlanations (SHAP)	A unified framework for interpreting model predictions.	Proposed
Amirian *et al.* [2021]	Non-specific	Feature response	A method for encoding and accounting for diversity within a class using inferred attributes in a zero-shot setting.	Combined methods
Zöller *et al.* [2023]	Non-specific	decision trees, LIME, PDP, ICE and Feature importance	XAutoML: An interactive visual analytics tool for explaining arbitrary AutoML optimization procedures and ML pipelines constructed by AutoML.	Combined methods
Narkar *et al.* [2021]	Non-specific	Feature importance, SHAP, interactive graphics	Model LineUpper: a model with the intent to support interactive model comparison for AutoML by integrating multiple Explainable AI (XAI) and visualization techniques	Combined methods
Garouani and Bouneffa [2023]	Non-specific	LIME, SHAP, ANCHORS and Saliency	An AutoML framework that enables users to understand the reasoning behind model recommendations and diagnose limitations using various explainable AI methods.	Combined methods
Moayeri *et al.* [2024]	Image processing	Attribute Inference, Nonlinear Prediction Consolidation and Attribute-based Explanations	A zero-shot method for enriching classes with open-ended attributes to boost zero-shot classification, utilizing generative language modeling along with a prediction consolidation step that prioritizes subpopulations most relevant to the image.	Combined methods
Bifarin and Fernández [2024]	Health domain	SHAP and Feature importance	A pipeline integrating AutoML with explainable AI (XAI) techniques to optimize metabolomics analysis	Combined methods
Mahmood *et al.* [2025]	Health domain	SHAP and LIME	A survey on predictive models for asthma outcomes, proposing an alternative leveraging AutoML and XAI, applying SHAP and LIME.	Survey
Yuan *et al.* [2024]	Health domain	Feature interaction and importance, data dimension, intrinsic model, knowledge distillation and rule extraction	A systematic review of AutoML interpretation systems in healthcare, illustrating techniques AutoML techniques used in the included publications accompanied by a real-world case study to demonstrate the advantages of AutoML over classic Ml.	Survey
Tian and Che [2024]	Health domain	feature importance, SHAP and LIME	A survey exploring the capabilities, limitations, and practical applications of six AutoML tools: Auto-sklearn, TPOT, H2O.ai, Google Cloud AutoML, Microsoft Azure AutoML, and Amazon SageMaker Autopilot.	Survey
Hasan *et al.* [2024]	Health domain	SHAP, LIME, Integrated Gradients (IG), Attention Mechanism (AM), and Counterfactual Analysis (CA)	A survey on integrating AutoML with XAI for diabetes prediction, evaluating SHAP, LIME, Integrated Gradients (IG), Attention Mechanism (AM), and Counterfactual Analysis (CA).	Survey
Karmaker S. et. al. [2021]	Non-specific	-	A survey on the state of the art in AutoML, proposing a level-based taxonomy for AutoML systems, defining each level according to its automation support	Survey and new taxonomy
Amirian *et al.* [2021]	Non-specific	-	A survey of recent advances, ideas, and practical industry applications of AutoML and neural network interpretability.	Survey

4 MH-AutoML

In this work, we present MH-AutoML, a domain-specific tool for Android malware detection that addresses transparency, interpretability, and lifecycle control limitations in existing AutoML solutions. Our tool implements a complete AutoML pipeline that handles data cleaning, feature engineering, algorithm selection, hyperparameter tuning, and optimization.

MH-AutoML advances beyond conventional AutoML by integrating experiment tracking, debugging, and interpretability features throughout the pipeline, enabling users to inspect internal operations rather than just receiving final performance metrics.

4.1 General Architecture and Tool Design

Figure 3 illustrates MH-AutoML's workflow, showing the key stages of the ML process, including data preprocessing, feature engineering, and model selection with tuning.

The initial Data Info stage performs dual exploratory analysis, examining both the computational environment (OS, hardware resources, and network metrics) and dataset characteristics (dimensionality, data types, class balance, and integrity measures).

During Preprocessing, we address data inconsistencies through normalization and transformation. The current version handles common issues like missing values, incorrect data types, redundant entries, and improperly filled fields through outlier removal, duplicate elimination, missing value treatment, and optional one-hot encoding. The tool generates visual artifacts such as data quality heatmaps (see examples available on GitHub [MH-AutoML, 2024]).

The Feature Engineering stage implements dimensionality reduction techniques including PCA (*Principal Component Analysis*) for feature extraction, ANOVA (*Analysis of Variance*) for statistical selection, and LASSO (*Least Absolute Shrinkage and Selection Operator*) for regularization. While PCA and ANOVA are common AutoML choices, LASSO was selected after thorough evaluation of over 20 feature selection methods across more than 10 heterogeneous datasets, showing superior generalization and robustness [Rocha *et al*., 2024].

For Model Selection, we consider the problem requirements and dataset characteristics when choosing algorithms. Our implementation uses `VotingClassifier` to combine predictions from `LightGBM`, `KNN`, `CatBoost`, `RandomForestClassifier`, `DecisionTreeClassifier`, and `ExtraTreesClassifier`. This ensemble approach improves robustness, and the architecture supports easy integration of additional algorithms like `SVM`.

MH-AutoML includes built-in interpretability methods across its model portfolio. While `DecisionTreeClassifier` offers inherent interpretability through its rule-based structure, ensemble methods like `CatBoost` and `LightGBM` provide feature importance analysis despite their greater complexity. `KNN` and `ExtraTreesClassifier` present intermediate interpretability levels, with the understanding that ensemble models may require more effort to fully interpret in feature-rich scenarios.

The Model Evaluation stage employs multiple metrics to assess performance:

- Precision: Correct positive predictions relative to total positive predictions
- Recall: Correct positive predictions relative to actual positives
- Accuracy: Correct predictions across all classes
- F1-score: Harmonic mean of precision and recall
- MCC (*Matthews Correlation Coefficient*): Binary classification quality metric
- Execution Time: Model training and evaluation duration
- ROC AUC (*Receiver Operating Characteristic Area Under Curve*): Class separation capability
- Precision-Recall Curve: Performance trade-off visualization
- CPU/RAM/Disk/Network Usage: Resource consumption metrics

During the tuning stage, we optimize the hyperparameters of each selected algorithm using the Optuna library, prioritizing Recall in order to improve malware detection effectiveness. This choice reflects the importance of minimizing false negatives in malware detection scenarios, where failing to identify malicious samples may have significant security implications.

To support model interpretability, we adopt a two-stage approach. In the pre-modeling phase, we perform a feature importance analysis to identify the most influential variables before model training, enabling a preliminary understanding of the dataset and its most relevant attributes. In the post-modeling phase, we apply LIME [Ribeiro *et al*., 2016] and SHAP [Lundberg and Lee, 2017] to interpret the predictions of the trained black-box models, providing insights into how individual features contribute to the model's decisions and improving the transparency of the predictive process.

MH-AutoML integrates with the open Machine Learning Operations (MLOps) platform[9] through MLflow[10], enabling pipeline lifecycle management. This includes model versioning, artifact storage (files, images, code), and experiment comparison - features not typically found in other MLflow-integrated tools. Additional pipeline artifacts and examples are available on GitHub[11].

5 Experimental Setup

5.1 Hardware and Software Environment

All experiments were executed on a dedicated server equipped with a dual-socket Intel Xeon E5649 processor (2×6 cores at 2.53 GHz) and 94 GB of RAM, running Linux Mint 20.3. The software environment was based on Python 3.10.12, with NumPy 1.23.5, Pandas 1.5.3, and scikit-learn 1.1.1 as core dependencies. Each AutoML framework was installed in an isolated environment following its official documentation in order to prevent dependency conflicts. No GPU acceleration was used, ensuring that all frameworks operated under identical hardware conditions.

[9] `https://ml-ops.org`

[10] `https://github.com/mlflow/mlflow`

[11] `https://github.com/SBSegSF24/MH-AutoML`

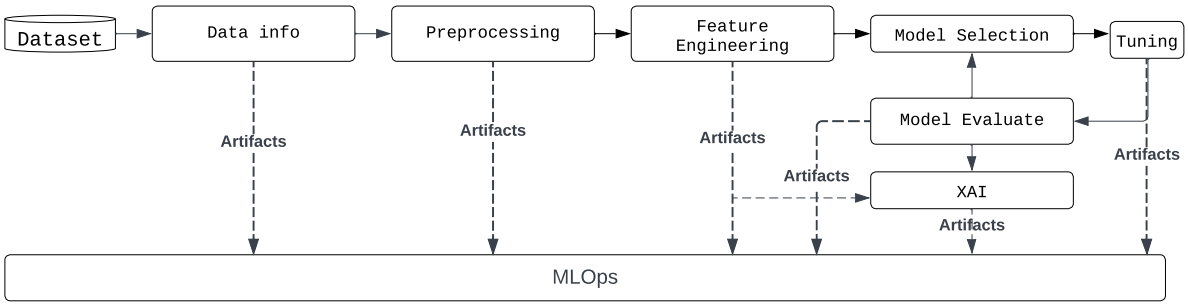

Figure 3. Pipeline of MH-AutoML.

To guarantee a fair comparison, each framework was executed using its default configuration, with the execution time limit adjusted only when necessary to allow the training process to complete. This design choice reflects a realistic deployment scenario in which practitioners adopt an AutoML tool with minimal manual configuration or parameter tuning.

5.2 Selection Criteria

For our comparative analysis, we selected seven established AutoML tools that represent the current state of open-source AutoML solutions: Auto-Sklearn [Feurer *et al*., 2015], 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], and MLJAR [Płońska and Płoński, 2021]. Our selection process followed four key criteria to ensure the relevance and quality of the tools included in our study:

- Complete implementation of all AutoML pipeline stages as defined by [Truong *et al*., 2019b]
- Open-source availability with permissive licenses
- Active maintenance with recent updates and community support
- Recognition in recent benchmarking studies [Ferreira *et al*., 2021; Karmaker S. et. al., 2021; Assolin *et al*., 2022; Bahri *et al*., 2022; Weerts *et al*., 2023]

5.3 Interpretability and Transparency Criteria

To systematically evaluate interpretability and transparency aspects, we developed a structured questionnaire based on established literature [Love *et al*., 2023; Arrieta *et al*., 2020; Mi *et al*., 2020; Carvalho *et al*., 2019; Brännström, 2023]. Our framework organizes questions into five key categories that address different dimensions of tool transparency and explainability.

The Functional Description category examines whether tools clearly document their pipeline stages, components, inputs, outputs, and execution processes. Statistical Analysis evaluates how tools present results and performance metrics, including class predictions, training history, feature importance, and visualization methods.

Algorithmic Transparency evaluates the clarity and availability of information related to models and hyperparameters, including aspects such as logging capabilities and strategies for handling data imbalance. The Interpretability category is divided into three complementary subcomponents: *pre-model* (explanations related to feature processing and data preparation), *in-model* (intrinsic interpretability mechanisms provided by the learning algorithm), and *post-model* (prediction analysis and model-agnostic explanation techniques).

Finally, the Internal Analysis category examines the ability of the tools to provide mechanisms for visualizing model structures and understanding their decision-making processes. Table 5 presents the complete evaluation framework adopted in this study, including the detailed scoring criteria for each dimension.

To enable systematic comparison, we developed a quantitative scoring model that assigns between 0 and 2 points to each question according to the quality of the implementation (Not applicable = 0, Partial = 1, Total = 2). Category scores are normalized to a 0–100% scale using the following formulation:

$$S = \frac{\sum_{i=1}^{N} P_i}{N \times W} \times 100$$

where S denotes the normalized score, P_i represents the score assigned to question i, N corresponds to the total number of questions in the category, and W denotes the maximum possible score per question (in this case, $W = 2$). This formulation enables fair comparison across tools while preserving scoring consistency across evaluation categories.

5.4 Datasets

The evaluation encompasses nine publicly available Android malware detection datasets that represent a diverse range of analysis approaches, feature engineering strategies, and class distributions (Table 6).

The datasets vary substantially in dimensionality (from 141 to 24,883 features) and cover five categories of static features: permissions (P), API calls (A), intents (I), system commands (S), and other characteristics (O). Class distributions range from approximately balanced sets (e.g., DefenseDroid with $\sim$6,000 samples per class) to highly imbalanced collections (e.g., AndroCrawl with 86,574 benign vs. 10,170 malicious samples, an 8.5:1 ratio). The datasets originate from established malware research: Adroit , AndroCrawl , Android Permissions , DefenseDroid , Drebin-215 , KronoDroid , and MH-100 .

This diversity serves two purposes. First, it tests whether AutoML tools generalise across different feature types

Table 5. Example of the Evaluation Model for Transparency and Interpretability of AutoML Tools.

Category	Question	Score	Score
Functional Description	Is it possible to identify the pipeline stages?	2	100
	Is it possible to identify the components used in each stage?	2	
	Are the inputs and outputs clear?	2	
	Is it clear how the process is executed in practice?	2	
Statistical Analysis	Are the model's predictions presented for the classes (malware and benign)?	2	100
	Is the history of data used to train and test the model maintained?	2	
	In feature selection, is the user informed about the most relevant features?	2	
	Are the model's performance metrics presented?	2	
	Are there ways to visualize the results (graphs, tables)?	2	
Algorithmic Transparency	Is it possible to identify which models are used?	2	100
	Is it possible to identify warnings, information, and errors recorded in the system logs?	2	
	Is it possible to identify the hyperparameters of the generated models?	2	
	Is it possible to identify if the data is balanced?	2	
Interpretability	Is the reason for dimensionality reduction explained? (Pre-Model)	2	100
	Are there domain-specific dimensionality reduction techniques? (Pre-Model)	2	
	Are global interpretability methods available?	2	
	Does the tool offer models with intrinsic interpretability? (In-Model)	2	
	Is it possible to evaluate the difference between predictions and actual values?	2	
	Are there model-agnostic interpretability methods? (Post-Model)	2	
Internal Analysis	Are there methods to visualize the model's structure? (Post-Model)	2	100

Table 6. Summary of Dataset Information.

Reference	*Dataset*	Features		Original Distribution	
		Qty.	Types	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 and Sezer [2018]	Drebin-215	215	A(73), P(113), S(6), I(23)	9476	5555
Guerra-Manzanares *et al*. [2021]	KronoDroid Emulator	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] Permissions, [A] API Calls, [I] Intents, [S] System Commands, [O] Others.

and data scales, from compact permission vectors to high-dimensional API call profiles. Second, the range of class imbalance ratios exposes framework sensitivity to skewed distributions — a critical concern in real-world malware detection, where benign applications vastly outnumber malicious ones.

5.5 Evaluation Scenarios

Each dataset is evaluated under two complementary scenarios. In the **original version**, the dataset is used with its native class distribution, preserving the natural imbalance present in the source collection. This scenario reflects the conditions typically encountered by practitioners when applying AutoML techniques to raw, unprocessed data.

In the **balanced version**, the class distribution is equalized prior to model training through random oversampling of the minority class. This configuration isolates the effect of class imbalance on framework performance and serves as a robustness diagnostic. Frameworks that maintain or improve their results under balancing are considered more robust, whereas performance degradation indicates sensitivity to distributional shifts.

The combination of nine datasets and two evaluation scenarios results in 18 distinct classification tasks. Each task is executed with 10 independent random seeds, produc-

ing $18 \times 10 = 180$ experimental runs per framework and $180 \times 8 = 1{,}440$ runs in total.

5.6 Evaluated Frameworks

Eight AutoML frameworks representing diverse search strategies were selected for evaluation:

1. **MH-AutoML**, the meta-heuristic-based framework proposed in this work.
2. **Auto-sklearn**, which applies Bayesian optimisation combined with meta-learning over scikit-learn pipelines.
3. **AutoGluon**, which relies on multi-layer stacking and ensemble selection strategies.
4. **HyperGBM**, focused on hyperparameter optimisation for gradient-boosted tree models.
5. **LightAutoML**, a lightweight pipeline designed for tabular data that incorporates automatic feature engineering.
6. **MLJAR**, which emphasises automated ensemble construction and provides an Explain mode to support interpretability.
7. **TPOT**, which performs pipeline optimisation using genetic programming.
8. **AutoPyTorch**, which combines neural architecture search with hyperparameter optimisation for deep learning pipelines.

This selection spans the major AutoML paradigms: Bayesian optimisation (Auto-sklearn), genetic/evolutionary search (TPOT, MH-AutoML), multi-fidelity neural search (AutoPyTorch), ensemble stacking (AutoGluon, MLJAR), gradient-boosted specialisation (HyperGBM), and hybrid approaches (LightAutoML). Including both traditional ML-based and deep learning-based frameworks enables a comprehensive assessment of which search strategies are best suited to the tabular feature spaces typical of Android malware detection.

5.7 Validation Methodology

Each experimental run follows a stratified holdout protocol: the dataset is randomly split into 80% training and 20% test sets, preserving the class distribution in both partitions. The split is controlled by a fixed random seed, ensuring that all eight frameworks are trained and evaluated on exactly the same data partition for each of the 10 seeds. Internal cross-validation and model selection are handled by each framework's default strategy (e.g., Auto-sklearn uses internal 10-fold CV; AutoGluon uses bagged k-fold), and are not imposed externally.

To quantify the stability of results, all metrics are computed independently for each of the 10 seeds. We report the *median* as the central tendency measure and the *standard deviation* as the dispersion measure across seeds. The median is preferred over the mean because it is robust to the occasional degenerate runs observed in certain frameworks (e.g., AutoPyTorch on *Android Permissions*), which would otherwise disproportionately influence the mean.

5.8 Evaluation Metrics

We evaluate each framework using five classification metrics and one efficiency metric:

- **Accuracy**: the proportion of correctly classified samples. While intuitive, accuracy can be misleading on imbalanced datasets, where a classifier predicting only the majority class achieves high accuracy without detecting any malware.
- **Precision**: the fraction of samples predicted as malicious that are truly malicious, i.e., Precision $= \mathrm{TP}/(\mathrm{TP}+\mathrm{FP})$. High precision indicates low false-positive rates.
- **Recall (Sensitivity)**: the fraction of actual malware samples correctly detected, i.e., Recall $= \mathrm{TP}/(\mathrm{TP}+\mathrm{FN})$. High recall is critical in security contexts where undetected malware carries severe consequences.
- **F1-Score**: the harmonic mean of precision and recall, $\mathrm{F1} = 2 \cdot \mathrm{Precision} \cdot \mathrm{Recall}/(\mathrm{Precision} + \mathrm{Recall})$. It provides a single balanced measure but, like accuracy, can be inflated by majority-class bias.
- **Matthews Correlation Coefficient (MCC)**: defined as

$$\mathrm{MCC} = \frac{\mathrm{TP} \cdot \mathrm{TN} - \mathrm{FP} \cdot \mathrm{FN}}{\sqrt{(\mathrm{TP}+\mathrm{FP})(\mathrm{TP}+\mathrm{FN})(\mathrm{TN}+\mathrm{FP})(\mathrm{TN}+\mathrm{FN})}}, \tag{1}$$

  where $\mathrm{MCC} \in [-1, +1]$, with $+1$ indicating perfect classification, 0 indicating random prediction, and -1 indicating complete disagreement. Unlike accuracy and F1, MCC accounts for all four confusion matrix quadrants and is considered the most informative single metric for binary classification on imbalanced data . We adopt MCC as the **primary evaluation metric** throughout this study.
- **Execution time**: the wall-clock time from the start of the AutoML search to the completion of model training (excluding data loading and evaluation). This metric captures the computational cost of each framework and is reported as the median across 10 seeds.

5.9 Statistical Testing

To determine whether the observed performance differences among the eight frameworks are statistically significant, we follow the protocol recommended by Demšar for comparing multiple classifiers over multiple datasets:

1. **Friedman test**: a non-parametric omnibus test that ranks the $k = 8$ frameworks independently on each of the $N = 9$ datasets and tests whether the average ranks differ significantly from the expected value under the null hypothesis of equal performance. The test statistic follows a χ^2 distribution with $k - 1 = 7$ degrees of freedom.
2. **Nemenyi post-hoc test**: if the Friedman test rejects the null hypothesis at significance level $\alpha = 0.05$, pairwise differences are assessed using the Nemenyi test. Two frameworks are considered significantly different if their average rank difference exceeds the critical distance

$$\mathrm{CD} = q_\alpha \sqrt{\frac{k(k+1)}{6N}}, \tag{2}$$

where q_α is the critical value of the Studentized range statistic divided by $\sqrt{2}$. For $k = 8$ and $N = 9$ at $\alpha = 0.05$, the critical distance is CD $= 3.500$.

Results are visualised using Critical Difference (CD) diagrams, in which frameworks are positioned along a horizontal axis according to their average rank and connected by thick bars when their difference is not statistically significant.

5.10 Pairwise Win/Tie/Loss Analysis

To complement the rank-based statistical tests, we compute pairwise Win/Tie/Loss (W/T/L) counts for each pair of frameworks (A, B) across all $N = 9$ datasets. For a given metric:

- *Win*: framework A achieves a strictly higher median value than framework B;
- *Tie*: both frameworks achieve equal median values (within floating-point tolerance);
- *Loss*: framework A achieves a strictly lower median value than framework B.

The *net score* of framework A against framework B is defined as the difference between the number of wins and the number of losses, that is, wins $-$ losses. These counts are aggregated into a $k \times k$ matrix in which rows represent frameworks and are ordered according to their total net score, placing the most dominant framework first.

This analysis complements the statistical results by providing an indication of practical significance that may not be captured by the Nemenyi test when the statistical power is limited, as in the case of $N = 9$. For instance, if a framework wins on 8 out of 9 datasets against a competitor, this outcome indicates clear practical dominance, even when the difference in average ranks does not exceed the critical distance.

5.11 Presentation of Results

The results are organised by metric and dataset version, with each combination analysed through three complementary visualisations:

1. **Performance heatmaps**: annotated colour-coded matrices with datasets as rows and frameworks as columns. Cell values represent the median performance (%) over 10 seeds. The colour scale ranges from light yellow (low values) to deep red (high values), providing immediate visual identification of per-dataset strengths and weaknesses.
2. **Critical Difference diagrams**: horizontal rank diagrams from the Friedman–Nemenyi procedure. Frameworks are positioned by average rank (best on the left, worst on the right), and thick connecting bars identify groups whose pairwise differences do not exceed the critical distance. The Friedman p-value, the number of datasets, and the CD value are annotated on each diagram.
3. **Win/Tie/Loss matrices**: 8×8 pairwise grids where each cell shows the W/T/L counts and net score for the row framework against the column framework. Background colour follows a diverging green–white–red scale, with green indicating row dominance and red indicating row inferiority.

Two additional visualisations provide global and multi-dimensional perspectives:

4. **Distribution boxplots** (Figure 16): a 3×2 FacetGrid displaying the distribution of dataset-level median values for precision, recall, and MCC across both the Original and Balanced versions. Boxes represent the interquartile range (IQR); whiskers extend to $1.5 \times$ IQR; and circles indicate outlier datasets. This view summarises each framework's central tendency, spread, and vulnerability to specific datasets.
5. **F1 vs. MCC bubble plot** (Figure 17): a 3×3 small-multiples grid (one subplot per dataset) plotting F1-Score on the x-axis against MCC on the y-axis, with bubble size proportional to execution time. This simultaneously visualises classification quality, metric agreement, and computational cost, enabling the identification of frameworks that achieve strong performance efficiently.

Finally, an aggregate W/T/L table (Table 7) consolidates the net scores from all four individual W/T/L analyses (MCC $\times$ Original/Balanced and Recall $\times$ Original/Balanced), providing a single global ranking of framework dominance across all evaluated conditions.

6 Results

This section presents the comparative evaluation of the eight AutoML frameworks across nine Android malware datasets, examining both the Original (imbalanced) and Balanced versions. We focus on two complementary metrics: *Matthews Correlation Coefficient* (MCC), which provides a balanced measure of classification quality even under class imbalance, and *Recall*, which captures the ability to detect malicious samples. Results are analysed through three complementary lenses: (i) performance heatmaps showing per-dataset median scores over 10 seeds, (ii) Critical Difference (CD) diagrams based on the Friedman–Nemenyi protocol Demšar [2006], and (iii) Win/Tie/Loss (W/T/L) count matrices quantifying pairwise dominance.

6.1 MCC on Original Datasets

Figure 4 presents the MCC performance of each framework across all nine datasets in their original class distributions. The heatmap reveals a wide spectrum of dataset difficulty. Drebin achieves the highest overall scores, with Auto-sklearn leading at 0.970 and six frameworks exceeding 0.960. Kronodroid Real and Androcrawl follow as relatively easy benchmarks, with most tools reaching 0.900–0.940 MCC. In contrast, Android Permissions stands out as extremely challenging: the best MCC is only 0.213 (MLJAR), and AutoPyTorch collapses to 0.098, indicating essentially random classification behaviour. Adroit represents a moderately difficult case (MCC 0.700–0.770), where MH-AutoML achieves the best result at 0.769.

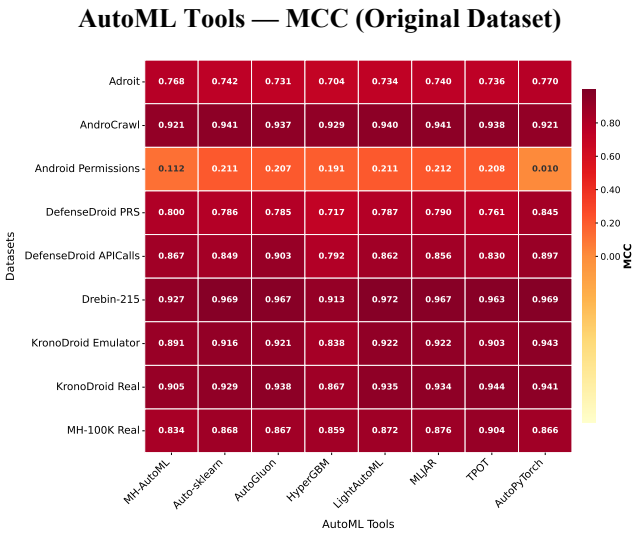

Figure 4. MCC performance across datasets for all evaluated AutoML tools — Original dataset version. Cell values represent mean MCC over 10 seeds.

To assess statistical significance, we applied the Friedman test followed by the Nemenyi post-hoc procedure. As shown in the CD diagram (Figure 5), the Friedman test confirms significant differences among the eight frameworks ($\chi^2 = 21.78$, $p = 0.0028$). MLJAR achieves the best average rank (2.78), followed by LightAutoML (3.44) and AutoPyTorch (3.56). HyperGBM ranks last (7.33) by a wide margin. The Nemenyi test at $\alpha = 0.05$ yields CD $= 3.500$, producing two overlapping cliques: one spanning MLJAR to MH-AutoML, and another from Auto-sklearn to HyperGBM. Only the pairwise differences between HyperGBM and the top three (MLJAR, LightAutoML, AutoPyTorch) reach statistical significance.

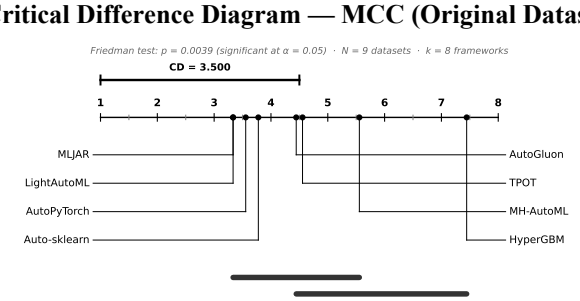

Figure 5. Critical Difference diagram for MCC — Original dataset version. Frameworks connected by a thick bar are not significantly different (Nemenyi, $\alpha = 0.05$).

The W/T/L matrix (Figure 6) complements the rank-based analysis by quantifying pairwise dominance across all nine datasets. MLJAR accumulates the highest net score (+31), winning 8 out of 9 datasets against both Auto-sklearn and AutoGluon. LightAutoML follows with +19, and AutoPyTorch with +17, reflecting strong MCC on datasets such as Kronodroid Emulador (0.942) and Defensedroid Prs (0.846). At the bottom, HyperGBM records a net of -51, losing all nine datasets against six of seven opponents. MH-AutoML finishes at -21, losing to most ensemble-based frameworks despite its first-place finish on Adroit.

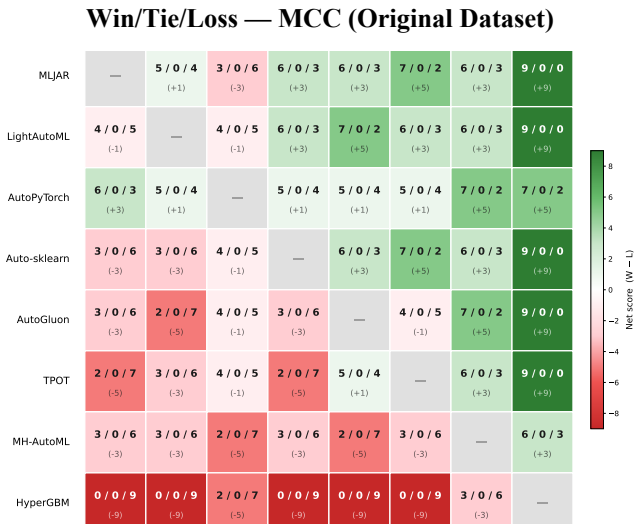

Figure 6. Win/Tie/Loss matrix for MCC — Original dataset version. Cell (A, B) shows wins/ties/losses of row framework A against column framework B across 9 datasets. Green = row dominance; red = column dominance; net score in parentheses.

6.2 MCC on Balanced Datasets

Figure 7 presents the MCC results obtained after applying class balancing. Two major changes are observed when compared with the Original version. First, the Android Permissions dataset shows improvement for all eight frameworks, with the best MCC increasing from 0.213 to 0.282 (TPOT). This result confirms that class imbalance was a relevant confounding factor for this dataset.

Second, AutoPyTorch exhibits severe performance degradation. On the Adroit dataset, MCC decreases from 0.766 to 0.113, corresponding to a reduction of 0.65, while on Drebin the score declines from 0.968 to 0.865. Overall, AutoPyTorch shows performance degradation on seven of the nine datasets, with improvement observed only on Android Permissions (+14.27 pp). MH-AutoML also degrades on six of the nine datasets, with the most notable reductions occurring on MH-100 (-5.5 pp) and Androcrawl (-4.7 pp).

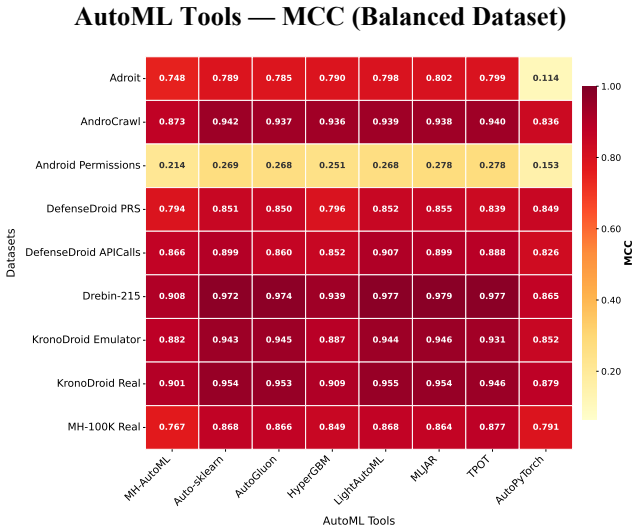

Figure 7. MCC performance across datasets for all evaluated AutoML tools — Balanced dataset version. Cell values represent mean MCC over 10 seeds.

The CD diagram (Figure 8) reveals substantially stronger separation among frameworks under class balancing. The

Friedman statistic nearly doubles ($\chi^2 = 43.81$, $p < 0.001$), confirming that balancing amplifies performance differences. Three overlapping cliques can be identified: (1) {MLJAR, LightAutoML, TPOT, Auto-sklearn, AutoGluon}, (2) {LightAutoML, TPOT, Auto-sklearn, AutoGluon, HyperGBM}, and (3) {AutoGluon, HyperGBM, MH-AutoML, AutoPyTorch}.

MH-AutoML (rank 7.00) and AutoPyTorch (rank 7.56) perform significantly worse than the top four frameworks. This result represents a notable inversion for AutoPyTorch, which ranked third under the Original dataset configuration.

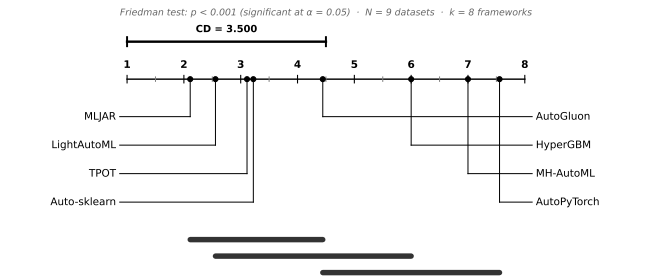

Figure 8. Critical Difference diagram for MCC — Balanced dataset version. Frameworks connected by a thick bar are not significantly different (Nemenyi, $\alpha = 0.05$).

The W/T/L matrix (Figure 9) confirms the polarisation. MLJAR achieves 9/0/0 clean sweeps against HyperGBM, MH-AutoML, and AutoPyTorch, with an aggregate net of +41. TPOT rises from a net of −1 (Original) to +23 (Balanced), ascending from rank 5–6 to rank 3, suggesting that its genetic programming search benefits from equalised class distributions. MH-AutoML's row is dominated by red: it records 0/0/9 against all five top-ranked frameworks. AutoPyTorch finishes last at −55.

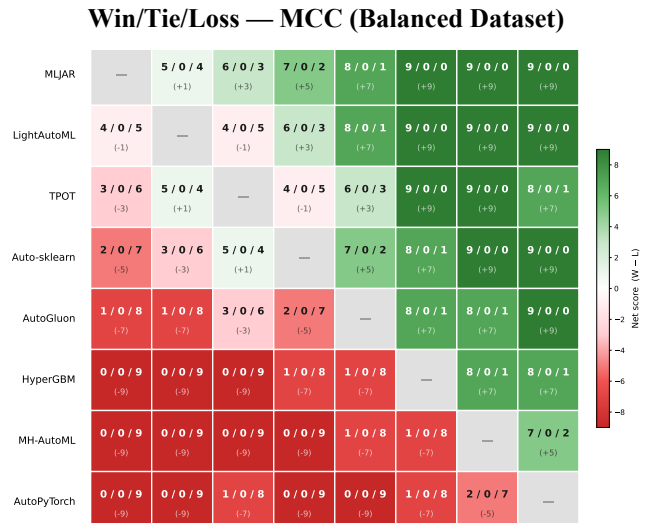

Figure 9. Win/Tie/Loss matrix for MCC — Balanced dataset version.

6.3 Recall on Original Datasets

Figure 10 presents the Recall heatmap for the Original version. A comparison with the MCC heatmap (Figure 4) reveals important divergences in framework behaviour. MH-AutoML, which ranked seventh according to MCC, achieves the best recall on three datasets: Androcrawl (0.985), MH-100 (0.971), and Adroit (0.905).

The difference between a recall rank of 4.67 and an MCC rank of 5.67 suggests that MH-AutoML strongly prioritises detection rate over precision. As a result, the generated models maximise the identification of malicious samples while producing a higher number of false positives.

The most striking cell in Figure 10 corresponds to AutoPyTorch on the Android Permissions dataset, which achieves 0.998 recall, the highest value observed in the entire study. However, this result corresponds to an MCC of only 0.098 (Figure 4). In this degenerate situation, the model predicts nearly all samples as belonging to the majority class, producing an artificially inflated recall while offering virtually no discriminative capability.

This example illustrates why recall alone is not a sufficient metric for evaluating malware classifiers and reinforces the importance of adopting MCC as the primary evaluation criterion.

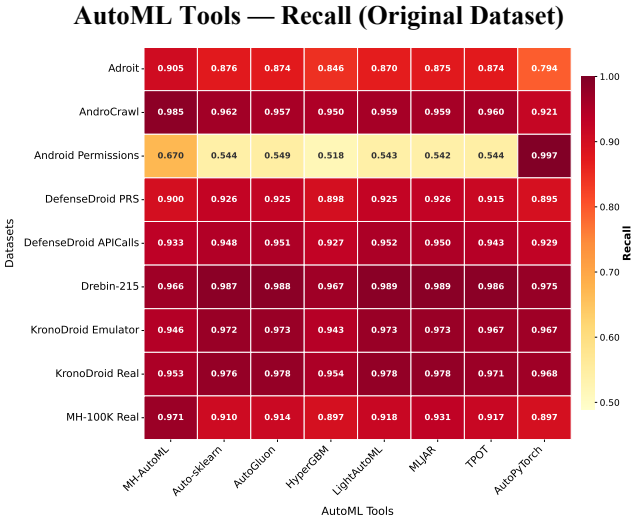

Figure 10. Recall performance across datasets for all evaluated AutoML tools — Original dataset version. Cell values represent mean recall over 10 seeds.

The CD diagram (Figure 11) indicates a significant Friedman test ($\chi^2 = 25.70$, $p < 0.001$). MLJAR and LightAutoML obtain the best average ranks (2.89 each), followed by Auto-sklearn (3.56) and AutoGluon (3.78). Two overlapping cliques can be identified among the top-performing frameworks.

AutoPyTorch (rank 6.56) appears in the lower tier. This position contrasts with its third-place ranking according to MCC and reflects its tendency to favour conservative decision thresholds that prioritise class separation while reducing the detection rate.

The W/T/L matrix (Figure 12) reveals that MLJAR and LightAutoML are perfectly tied at a net of +29 each (46 wins, 17 losses out of 63 pairwise comparisons). This parity is notable because on MCC, MLJAR leads LightAutoML by +12 net (31 vs. 19). The discrepancy arises because MCC penalises false positives more heavily: LightAutoML achieves comparable recall but with slightly better precision on certain datasets, producing equivalent recall dominance despite

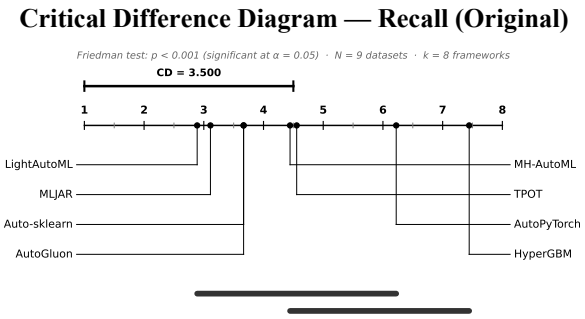

Figure 11. Critical Difference diagram for Recall — Original version.

lower MCC dominance. At the bottom, HyperGBM records a net of -45, losing all nine datasets (0/0/9) against the top five frameworks.

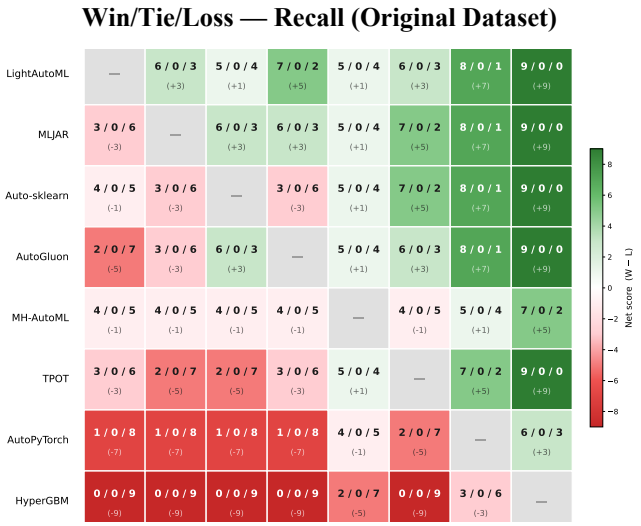

Figure 12. Win/Tie/Loss matrix for Recall — Original version.

6.4 Recall on Balanced Datasets

Figure 13 reveals MLJAR's strongest performance across all analyses. MLJAR ranks 1st on five of nine datasets (Android Permissions, Defensedroid Prs, Drebin, Kronodroid Emulador, and Kronodroid Real) and 2nd on three others (Adroit, Androcrawl, Kronodroid Real). Its average rank of 1.89 is the most dominant position observed for any framework in any metric–version combination in this study.

AutoPyTorch's collapse is again visible. On the Adroit dataset, recall decreases from 0.795 (Original) to 0.225 (Balanced), which corresponds to the most severe single-dataset degradation observed in the entire benchmark. This reduction is consistent across all 10 seeds, confirming that the failure is systematic rather than attributable to random initialisation.

The CD diagram (Figure 14) confirms a significant Friedman test ($\chi^2 = 30.93, p < 0.001$). MLJAR leads at rank 1.89 with a gap of more than one full rank position over LightAutoML (3.00). Two cliques emerge: {MLJAR, LightAutoML, AutoGluon, Auto-sklearn} and {AutoGluon, Auto-sklearn, TPOT, MH-AutoML, HyperGBM}. AutoPyTorch (rank 6.89) falls outside both cliques, confirming its significant inferiority to the top group.

The W/T/L matrix (Figure 15) confirms the dominance of MLJAR, which achieves a net score of $+47$, the highest net

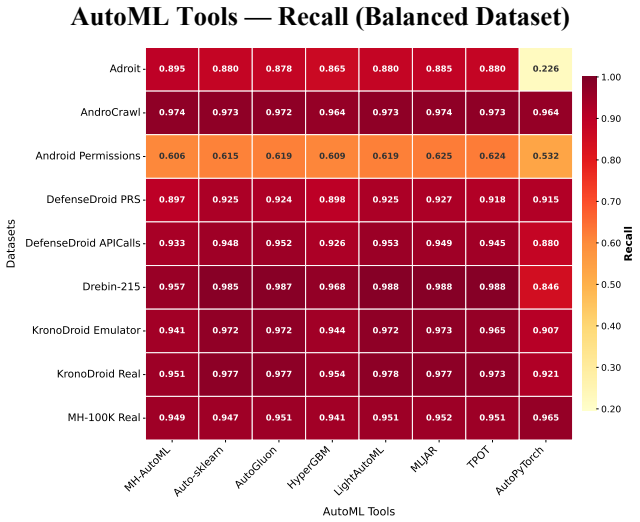

Figure 13. Recall performance across datasets for all evaluated AutoML tools — Balanced version. Cell values represent mean recall over 10 seeds.

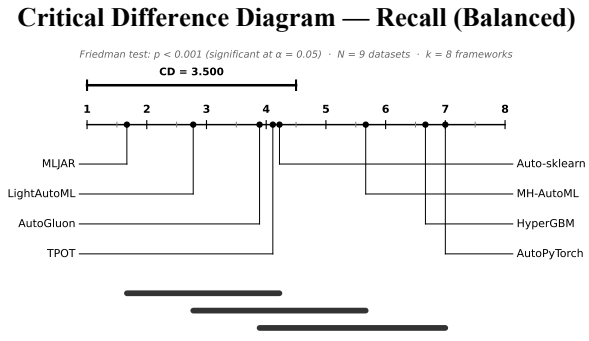

Figure 14. Critical Difference diagram for Recall — Balanced version.

value observed for any framework across the four analyses. MLJAR records 9/0/0 results against both HyperGBM and TPOT, and 8/0/1 against LightAutoML, AutoGluon, Auto-sklearn, and AutoPyTorch. In contrast, AutoPyTorch finishes with a net score of -43, losing 8 out of 9 datasets against each of the top-five frameworks.

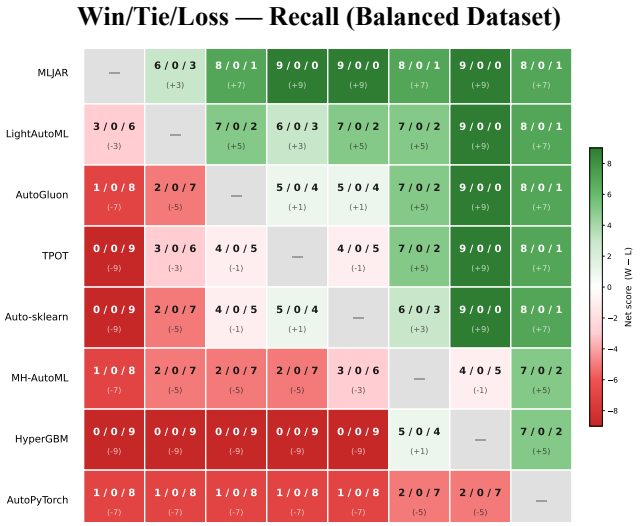

Figure 15. Win/Tie/Loss matrix for Recall — Balanced version.

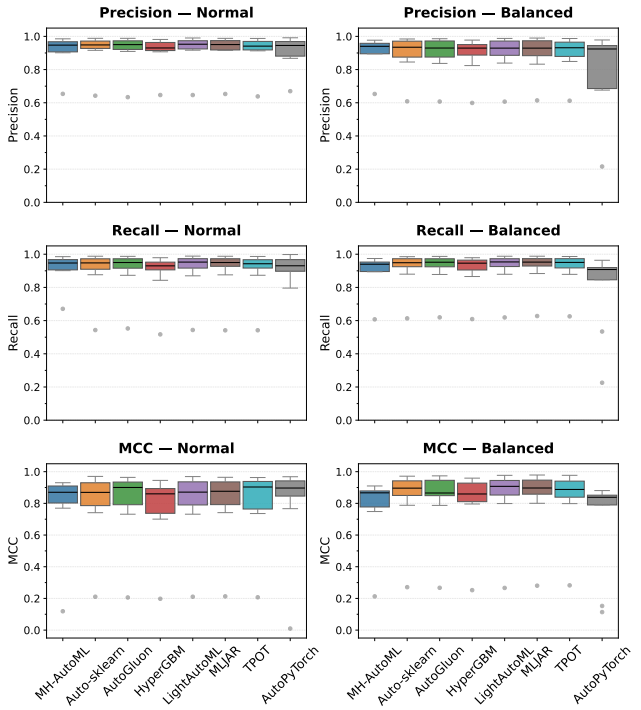

Figure 16. Distribution of precision, recall and MCC across all datasets and seeds for each framework. Boxes represent the IQR; whiskers extend to 1.5 × IQR; circles indicate outliers. Left column: Original version; right column: Balanced version.

6.5 Global Distribution of Performance Metrics

While the heatmaps and W/T/L matrices focus on per-dataset comparisons, Figure 16 provides a complementary global view by aggregating each framework's performance across all nine datasets. Each box summarises the distribution of dataset-level median values, revealing both the central tendency and the spread of results.

6.5.1 Precision

In the Original version (top-left panel), all eight frameworks exhibit similar median precision, clustered between 0.93 and 0.95, with compact interquartile ranges (IQR). Every framework presents a single low outlier near 0.65, corresponding to the *Android Permissions* dataset, which uniformly depresses precision regardless of the AutoML approach. AutoPyTorch's box is slightly wider than the others, reflecting greater variability across datasets even before class balancing.

In the Balanced version (top-right panel), the boxes for the top six frameworks remain largely unchanged. AutoPyTorch, however, shows a markedly wider box and a pronounced low outlier near 0.22, corresponding to its degenerate behaviour on *Adroit Balanced*. This visible expansion of AutoPyTorch's IQR from Original to Balanced confirms the instability identified in the per-dataset analysis (Section A.3.3).

6.5.2 Recall

The recall panels (middle row) reveal an important distinction. In the Original version, most frameworks present tightly concentrated distributions above 0.90, with lower outliers around 0.54 primarily driven by the *Android Permissions* dataset. The lower outlier of MH-AutoML is noticeably higher (0.67) than the cluster observed for the other frameworks (≈ 0.54), reflecting its stronger recall on *Android Permissions* at the expense of MCC. AutoPyTorch is the only framework that does not exhibit a visible low outlier in the Original recall panel because its degenerate prediction on *Android Permissions* (recall = 0.998) shifts the value upward rather than downward.

In the Balanced version (middle-right), AutoPyTorch presents two distinct low outliers: one at 0.23 on *Adroit* and another at 0.53 on *Android Permissions*. No other framework exhibits more than a single outlier, which confirms that AutoPyTorch's failures under balancing are not restricted to a single dataset.

6.5.3 MCC

The MCC panels (bottom row) display the widest spread, with boxes spanning approximately 0.77–0.94 for most frameworks, reflecting the intrinsic difficulty range of the benchmark suite. HyperGBM presents a visibly lower and wider box in the Original version, with its lower whisker extending below 0.70, which is consistent with its last-place position in the CD diagram. In the Balanced version, HyperGBM shows an increase in the median and a narrower box, suggesting that class balancing partially stabilises its behaviour.

AutoPyTorch exhibits the most pronounced shift between the two versions. In the Original panel, its box remains competitive, with a median around 0.90 and a single extreme low outlier near 0.01 (*Android Permissions*). In the Balanced panel, a second low outlier appears near 0.11 (*Adroit*), and the box shifts downward. Among the top-performing frameworks, MLJAR, LightAutoML, and Auto-sklearn display the highest medians and the most compact boxes in both versions,

which corroborates their consistently low variance across seeds and datasets.

Collectively, Figure 16 confirms two central findings. First, the choice of evaluation metric has a substantial impact on the interpretation of results. Precision boxplots appear nearly indistinguishable across frameworks, whereas MCC boxplots reveal clear separation, reinforcing MCC as the more informative criterion for performance assessment.

Second, the transition from the Original to the Balanced version functions as a visual stress test. Robust frameworks such as MLJAR and LightAutoML maintain compact distributions with high medians in both columns, whereas more fragile frameworks such as AutoPyTorch and HyperGBM display wider boxes, lower medians, or additional outliers.

6.6 F1-Score vs. MCC Trade-off with Execution Time

Figure 17 presents a multi-dimensional view by plotting F1-Score against MCC for each framework on each dataset, with bubble size encoding execution time. This visualisation simultaneously addresses three questions: how well do frameworks discriminate between classes (MCC, y-axis), how well do they balance precision and recall (F1, x-axis), and what computational cost do they incur (bubble size)?

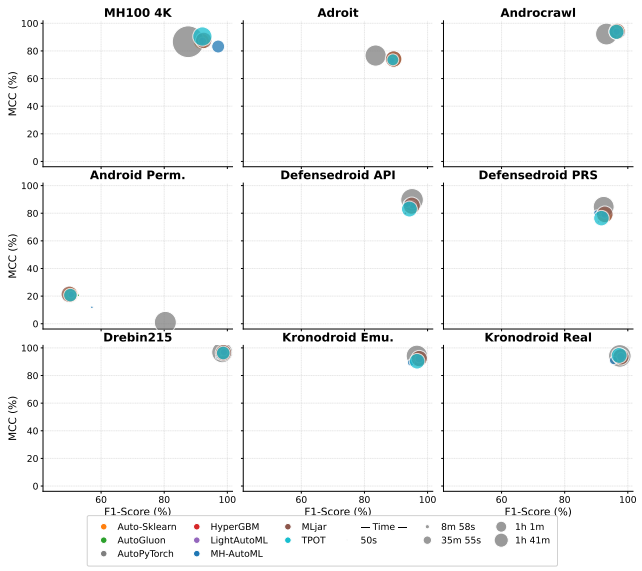

Figure 17. F1-Score vs. MCC (Original version) for each framework across the nine datasets. Each bubble represents one framework on one dataset; bubble size is proportional to median execution time. Larger bubbles indicate longer runtimes.

Bubble size encodes the computational cost, revealing a clear three-tier time hierarchy. HyperGBM consistently shows the smallest bubbles (15–50 seconds), followed by MH-AutoML and AutoGluon with small-to-medium bubbles (50 seconds to 15 minutes). Auto-sklearn, MLJAR, and TPOT produce medium-to-large bubbles (30 minutes to just over 1 hour), while AutoPyTorch generates the largest bubbles across all datasets (1h 41m to 3h 55m).

On the easy datasets (*Drebin*, *Kronodroid*), the bubble sizes vary by an order of magnitude while the performance positions are nearly identical, demonstrating that expensive frameworks provide no benefit when the classification task is inherently easy. On the moderate datasets, the trade-off becomes more meaningful: AutoGluon (small bubbles, typically under 15 minutes) achieves positions comparable to MLJAR and Auto-sklearn (large bubbles, approximately one hour), making it the most efficient choice for comparable quality. LightAutoML offers a similar trade-off, with medium bubbles (9–17 minutes) positioned consistently near the top of each cluster.

On *MH-100*, the bubble corresponding to AutoPyTorch is both the largest (3h 55m) and positioned among the lowest in terms of MCC (0.867), whereas TPOT achieves the highest MCC (0.904) with a substantially smaller bubble (1h 26m). This result illustrates that the framework with the longest execution time does not necessarily produce the best performance, which is an important practical consideration when deploying AutoML pipelines at scale.

6.7 Cross-Metric Synthesis

Table 7 aggregates the net W/T/L scores across all four analyses.

Table 7. Aggregate net W/T/L scores across all four analyses (MCC and Recall × Original and Balanced). Higher totals indicate greater overall dominance across 9 datasets × 7 opponents × 4 settings = 252 pairwise comparisons per framework.

Framework	MCC-Orig	MCC-Bal	Rec-Orig	Rec-Bal	Total
MLJAR	+31	+41	+29	+47	**+148**
LightAutoML	+19	+33	+29	+27	+108
Auto-sklearn	+7	+21	+17	+5	+50
AutoGluon	−1	+7	+13	+15	+34
TPOT	−1	+23	−3	−1	+18
MH-AutoML	−21	−45	−3	−11	−80
AutoPyTorch	+17	−55	−37	−43	−118
HyperGBM	−51	−25	−45	−39	−160

Three distinct tiers emerge. **Tier 1**: MLJAR (+148) and LightAutoML (+108) maintain positive net scores across all four evaluation settings, with MLJAR achieving a total that is 37% higher than that of LightAutoML. This gap is primarily driven by MLJAR's superior Recall performance on Balanced datasets (+47 versus +27).

Tier 2: Auto-sklearn (+50), AutoGluon (+34), and TPOT (+18) form a competitive middle group. TPOT's positive aggregate is largely attributable to its MCC performance on the Balanced setting (+23), where its genetic programming strategy appears to benefit from the equalised class distributions.

Tier 3: MH-AutoML (−80), AutoPyTorch (−118), and HyperGBM (−160) occupy the lower ranks. The aggregate score of AutoPyTorch is particularly informative, since its positive MCC result in the Original setting (+17) is outweighed by substantial deficits in the other three evaluation scenarios.

The cross-metric comparison reveals systematic biases in specific frameworks. MH-AutoML exhibits a much larger deficit in MCC (−21) than in Recall (−3) on the Original datasets, indicating a tendency to over-predict the positive class. This behaviour maximises malware detection while increasing the number of false positives. AutoPyTorch shows the opposite pattern on the Original datasets, with strong MCC performance (+17) but weak Recall (−37), which reflects the use of conservative decision thresholds that prioritise discrimination over sensitivity. When class balancing is applied, both strategies deteriorate.

In contrast, MLJAR and LightAutoML maintain consistent performance across both evaluation metrics. On the Original datasets they obtain +31/+29 and +19/+29 for MCC and Recall, respectively. On the Balanced datasets they achieve +41/+47 and +33/+27, confirming their robustness with respect to both metric choice and distributional changes.

Finally, the analysis indicates that class balancing functions as a diagnostic stress test. The Friedman χ^2 statistic for MCC increases substantially from 21.78 (Original) to 43.81 (Balanced), and the number of significant Nemenyi pairwise differences rises from 3 to 9. Frameworks that rely on implicit assumptions about class priors, particularly AutoPyTorch and MH-AutoML, exhibit noticeable performance degradation, whereas ensemble-based approaches (MLJAR, LightAutoML, Auto-sklearn, and TPOT) maintain or even improve their results.

6.8 Summary of Key Findings

(1) MLJAR and LightAutoML constitute a clear top tier. MLJAR achieved the highest aggregate W/T/L net score ($+148$) across all four metric–version analyses, followed by LightAutoML ($+108$). MLJAR ranked first on MCC for both the Original (rank 2.78) and Balanced (rank 2.22) versions, while LightAutoML tied for first on Recall Original (rank 2.89) and maintained second place across all remaining analyses. These two frameworks were the only ones with positive net scores on every individual W/T/L table, demonstrating consistent superiority across metrics and dataset versions.

(2) LightAutoML offers the best efficiency–performance trade-off. While MLJAR and LightAutoML achieve comparable classification quality, LightAutoML does so in 9–17 minutes per dataset compared to MLJAR's approximately one hour. For practitioners operating under computational constraints or requiring rapid iteration, LightAutoML provides near-optimal performance at a fraction of the cost. AutoGluon (1–15 minutes) also offers competitive efficiency, though at a slightly lower performance level.

(3) MCC is the essential metric for malware detection evaluation. The study provides compelling evidence that accuracy, recall, and F1-Score can produce misleading evaluations on imbalanced malware datasets. AutoPyTorch on *Android Permissions* achieved recall of 0.998 and F1 of 0.803 while producing an MCC of only 0.098, indicating degenerate majority-class prediction. This case demonstrates that MCC, by accounting for all four confusion matrix quadrants, is the only single metric that reliably detects such failures. We recommend that future Android malware classification studies adopt MCC as their primary evaluation criterion.

(4) Class balancing acts as a robustness stress test. The transition from Original to Balanced datasets doubled the Friedman chi-square for MCC (from 21.78 to 43.81) and tripled the number of statistically significant pairwise differences. Six of eight frameworks improved their average MCC under balancing (by 2–4 percentage points), confirming that class imbalance is a substantial confound. However, two frameworks degraded: AutoPyTorch suffered a catastrophic 65-percentage-point MCC collapse on *Adroit*, and MH-AutoML declined on six of nine datasets. These results suggest that class balancing should be adopted not only to improve classification but also as a diagnostic tool to identify fragile AutoML configurations.

(5) No single framework dominates all datasets. Despite MLJAR's overall superiority, MH-AutoML achieved the best MCC on *Adroit* (Original), TPOT led on *Kronodroid Real* and *MH-100* (Original), AutoGluon topped *Defensedroid API Calls* (Original), and Auto-sklearn achieved the highest MCC on *Drebin* (Original). This dataset-dependent variation motivates meta-learning and per-dataset framework selection strategies as a promising direction for future research.

(6) HyperGBM and AutoPyTorch carry significant deployment risks. HyperGBM accumulated the worst aggregate net (-160), losing all nine datasets against six of seven competitors on MCC Original, and exhibited the highest seed-to-seed variance (average $\sigma_{\text{MCC}} = 0.045$, ten times higher than Auto-sklearn's 0.004). AutoPyTorch's rank swing from 3rd (MCC Original) to 8th (MCC Balanced) represents the largest instability of any framework across all analyses. These findings advise against deploying either tool in production malware detection systems without extensive validation.

(7) MH-AutoML exhibits a systematic recall bias. The proposed framework achieved competitive recall (net -3 on Original, compared with MCC net -21) by aggressively optimising the detection rate at the expense of precision. On *MH-100*, MH-AutoML ranked first in recall but eighth in MCC, which corresponds to a discrepancy of seven ranking positions. Although this behaviour may be acceptable in triage scenarios where the cost of missing malware exceeds the cost of false alarms, it limits the applicability of MH-AutoML in contexts that require balanced discrimination.

6.9 Limitations

This study presents several limitations that should be considered when interpreting the results. These limitations relate to the size of the benchmark, the configuration strategy adopted for the evaluated frameworks, the type of features used in the experiments, and the scope of the classification task.

First, the benchmark includes nine datasets, which restricts the statistical power of rank-based post-hoc tests. The Nemenyi critical distance of $\text{CD} = 3.500$ indicates that meaningful practical differences may not reach statistical significance under this setting. For example, Auto-sklearn's aggregate net score of $+50$ compared with TPOT's $+18$ represents a clear practical advantage, yet the difference cannot be formally confirmed at conventional significance levels. Expanding the benchmark with additional datasets and greater diversity would increase the discriminative power of the statistical analysis.

Second, all frameworks were evaluated using their default configurations. This choice reflects a realistic deployment scenario in which practitioners rely on the standard settings provided by each framework and perform minimal manual configuration. However, default settings may not always align with the characteristics of Android malware datasets, which often contain high-dimensional and sparse binary features. A complementary study involving framework-specific hyperparameter tuning could potentially alter the relative rankings observed in this work.

Third, the evaluation relies exclusively on static features

extracted from Android applications. Although static analysis is widely used in malware detection research, the inclusion of dynamic analysis features could change the complexity of the learning task. Runtime API call sequences, network traffic patterns, or system call traces may capture behavioural characteristics that are not observable through static analysis alone, which could influence the comparative performance of the evaluated frameworks.

Finally, the study focuses only on the binary classification setting, distinguishing between benign and malicious applications. In practice, malware analysis often requires multi-class classification in order to identify malware families or behavioural clusters. Such tasks are relevant for threat characterisation and incident response and may favour different modelling strategies or AutoML frameworks than those observed in the binary detection scenario.

7 Threats to Validity

We organize the threats to the validity of this study according to the four canonical categories established in empirical software engineering research: internal validity, external validity, construct validity, and conclusion validity. Where applicable, we identify concrete opportunities for future work that directly address each threat.

7.1 Internal Validity

The most prominent internal threat concerns data partition variability and seed-induced bias. Although we mitigated this risk by executing each framework across ten distinct random seeds and reporting median values with standard deviations, residual sensitivity to initialization remains. In particular, HyperGBM exhibited the highest coefficient of variation across seeds and datasets, suggesting that its reported performance is more dependent on random initialization than that of more stable frameworks. Future work should provide a more formal quantification of this sensitivity, for instance, through bootstrap confidence intervals or permutation-based significance tests, to establish tighter robustness guaranties for all compared frameworks.

A second internal threat arises from the class imbalance treatment applied in the balanced experimental condition. Random undersampling, while straightforward to reproduce, introduces information loss and may bias results toward the distribution of the sampled majority subset. The choice of balancing strategy is itself a confounding variable: alternative approaches such as SMOTE, class-weighted loss functions, or ensemble-based resampling could yield substantially different outcomes for the same framework, and the current design does not isolate these effects. Future investigations should adopt a factorial experimental design in which the balancing strategy is treated as an independent variable, enabling a principled assessment of its interaction with framework architecture.

A third internal threat concerns the configuration homogeneity assumption. All frameworks were evaluated under their default or minimally adjusted settings to ensure a fair baseline comparison. However, this choice means that performance differences may partly reflect the quality of each framework's defaults rather than its fundamental capabilities. Frameworks such as Auto-Sklearn, TPOT, and AutoPyTorch, which rely on computationally intensive meta-learning and genetic optimization strategies, may achieve substantially higher performance under extended time budgets. Future work should complement the default-configuration evaluation with a resource-controlled comparison in which all frameworks are allocated identical wall-clock budgets.

7.2 External Validity

Our evaluation is grounded in nine Android malware datasets covering diverse feature types (permissions, API calls, intents, and system commands), a wide range of sizes, and varying class imbalance ratios. Nevertheless, all datasets reflect malware families and collection periods available at the time of study. The generalizability of the results to previously unseen malware families, particularly those employing advanced evasion techniques such as polymorphism, dynamic code loading, reflection abuse, or anti-analysis obfuscation, cannot be guaranteed. As the Android threat landscape evolves continuously, periodic re-evaluation of all compared frameworks on updated datasets is a necessary step toward sustained external validity. Integrating a continuous retraining and re-evaluation pipeline into MH-AutoML represents a natural direction for future work.

Beyond dataset scope, all experiments were conducted on a single hardware configuration (Intel Xeon E5649, 94 GB RAM, Linux Mint 20.3). Execution time measurements and memory utilization patterns may differ substantially on commodity hardware, GPU-accelerated infrastructure, or cloud-based deployment environments. Furthermore, the current study addresses binary classification exclusively, distinguishing malware from benign applications. Extending MH-AutoML to multi-class settings, such as per-family malware classification, and to other cybersecurity domains would substantially broaden the external validity of the framework and represents a primary avenue for future development.

7.3 Construct Validity

The primary construct threat concerns the operationalization of interpretability and transparency. We assess these constructs through a structured questionnaire of our own design, grounded in established literature and organized across five theoretically motivated dimensions. However, the instrument remains a proxy for the underlying constructs, and it has not yet been validated through inter-rater reliability studies or user experiments with practicing security analysts. Scores assigned to partial implementations (value 1 in the 0–2 scale) are inherently judgmental, and a different set of evaluators could plausibly assign different scores to borderline cases. Moreover, transparency dimensions such as *Algorithmic Transparency* and *Internal Analysis* may carry different weights depending on the user profile: a security researcher optimizing model behavior has different interpretability needs than an operational analyst assessing deployment risk. Our uniform weighting scheme does not capture these differences. Future work should involve independent evaluators and domain experts to establish inter-rater reliability, and should

explore adaptive weighting schemes that reflect the priorities of distinct practitioner profiles, thereby improving both the reliability and the ecological validity of the scoring instrument.

A secondary construct threat concerns metric asymmetry across frameworks in the original (imbalanced) dataset condition. Because MH-AutoML computes recall as a class-averaged mean by default, rather than applying class-weighted adjustments or internal oversampling, its reported recall values can be misleading when class distributions are highly skewed. As discussed in previous sections, MH-AutoML's 0.537 recall on the Android Permissions dataset reflects this aggregation artifact rather than a failure of the malware-class classifier, which achieved 0.960 recall on that class in isolation. Frameworks that transparently disclose their internal resampling and weighting strategies provide a clearer basis for comparison; the lack of consistent documentation of these pipeline decisions in several third-party tools is itself a construct threat, making it difficult to attribute performance differences to framework design rather than to implicit preprocessing choices. Standardizing metric reporting protocols across AutoML benchmarks is an open methodological challenge that future comparative studies should address explicitly.

7.4 Conclusion Validity

To support statistically grounded conclusions, we employed the non-parametric Friedman test with Holm post-hoc corrections across 90 observations per framework per experimental condition (9 datasets $\times$ 10 seeds). This design provides adequate statistical power for detecting systematic performance differences. However, the Friedman test ranks frameworks globally across all datasets, which may obscure dataset-specific advantages. MH-AutoML, for instance, leads in precision on two to three datasets per condition, a finding that is structurally diluted in the global ranking. Practitioners with requirements tied to specific dataset characteristics, such as large feature spaces, extreme imbalance, or behavioral feature sets, should therefore examine per-dataset results alongside aggregate rankings when selecting a framework.

Finally, the primary focus of this study on binary Android malware classification naturally constrains the scope of the conclusions drawn. Future developments for MH-AutoML will concentrate on deepening the framework's transparency and interpretability within this scope while progressively extending it. The principal planned contribution is a *continuous interpretability indicator*: a dynamic, quantitative metric that evaluates the stability and quality of model explanations, such as those produced by LIME and SHAP, across retraining cycles over time. This indicator would complement the current qualitative assessment framework, providing analysts with an objective and monitorable measure of explanation drift and model trustworthiness, which is particularly critical in production environments where malware families evolve and models must be retrained periodically.

8 Conclusion

Our comprehensive evaluation compares MH-AutoML with seven widely used AutoML frameworks (AutoGluon, Auto-Sklearn, TPOT, AutoPyTorch, MLJAR, HyperGBM, and LightAutoML) in the context of Android malware detection. The analysis highlights clear differences in transparency, interpretability, execution efficiency, and predictive performance, revealing important strengths and trade-offs among the evaluated tools.

MH-AutoML stands out primarily due to its advanced interpretability and transparency capabilities. The framework incorporates specialised feature selection methods and model-specific analysis tailored to Android malware classification. In addition, it integrates experiment tracking and versioning through MLflow, enabling full traceability of the AutoML pipeline. This functionality supports detailed inspection of model development stages and facilitates reproducibility, which is particularly valuable in research and debugging scenarios. Although its current version produces competitive results compared with established tools, there remains room for further optimisation in future versions.

The analysis of transparency and interpretability features (Figure 18) shows that both MH-AutoML and MLJAR provide user-friendly environments suitable for non-expert users. MH-AutoML further extends this capability through comprehensive experiment tracking using MLflow, which provides detailed visibility into the model training and optimisation process. Even though these tools do not always achieve the highest detection scores, they produce stable and consistent results across the evaluated datasets while maintaining reasonable execution times. This combination of usability, transparency, and stability makes them attractive options for users with limited machine learning expertise.

Execution efficiency varies considerably across frameworks. HyperGBM and AutoGluon emerge as the fastest solutions in our experiments. HyperGBM achieves this efficiency through several optimisation strategies, including early stopping mechanisms that reduce training time and mitigate overfitting. AutoGluon adopts a different strategy by using fixed hyperparameters across its ensemble of models, thereby avoiding the computational overhead associated with extensive hyperparameter optimisation while still maintaining competitive predictive performance.

With respect to classification effectiveness, AutoGluon and LightAutoML consistently achieve the highest accuracy-oriented metrics across the evaluated datasets. LightAutoML appears as the most balanced framework overall, combining strong predictive performance with moderate execution times. AutoGluon, HyperGBM, MH-AutoML, and MLJAR also demonstrate favourable trade-offs between performance and computational cost, although each framework emphasises different aspects of the optimisation process. In contrast, AutoPyTorch, Auto-Sklearn, and TPOT appear less competitive in the current benchmark, particularly when execution time and predictive performance are considered simultaneously.

These findings indicate that the choice of AutoML framework should be guided by the specific requirements of the target application:

- For maximum interpretability and transparency: MH-AutoML or MLJAR;
- For fastest execution: HyperGBM or AutoGluon;
- For best classification performance: AutoGluon or LightAutoML;
- For balanced overall performance: LightAutoML.

Overall, the results show that no single framework dominates all evaluation dimensions. Nevertheless, the current ecosystem of AutoML solutions provides several viable alternatives for Android malware detection. Each tool presents distinct advantages that practitioners can leverage depending on the operational constraints, interpretability requirements, and performance objectives of their specific use case.

Declarations

Authors' Contributions

Joner Assolin and Diego Kreutz contributed to the conception and design of this study. Joner Assolin and Gabriel Canto conducted the experiments. Diego Kreutz, Hendrio Bragança, Eduardo Feitosa, Angelo Nogueira and Vanderson Rocha provided critical revisions, supervision, and feedback. Angelo Nogueira was one of the primary authors of Section 3. Joner Assolin, Gabriel Canto, and Diego Kreutz were the main contributors and writers of the manuscript. All authors reviewed and approved the final version.

Competing interests

The authors declare that they have no competing interests.

Funding

This research was financed by multiple agencies and through public-private partnerships. It was partially funded as stipulated in Articles 21 and 22 of Decree No. 10.521/2020, under Federal Law No. 8.387/1991, through agreement No. 003/2021, signed between the Institute of Computing of the Federal University of Amazonas (ICOMP/UFAM), Flextronics da Amazônia Ltda., and Motorola Mobility Comércio de Produtos Eletrônicos Ltda.

This work was also supported by the Coordination for the Improvement of Higher Education Personnel – Brazil (CAPES) – Financing Code 001, and partially supported by the Amazonas State Research Support Foundation (FAPEAM) through the POS-GRAD project 2024/2025. Additionally, this research was partially funded by the Rio Grande do Sul State Research Support Foundation (FAPERGS) through grant agreements 24/2551-0001368-7 and 24/2551-0000726-1. Finally, we acknowledge the support of the National Council for Scientific and Technological Development – Brazil (CNPq) through process No. 409743/2025-9.

Availability of data and materials

The source code and datasets analyzed in this study are publicly available in the MH-FSF GitHub repository: `https://github.com/SBSegSF24/MH-FSF`.

References

Amirian, M., Tuggener, L., Chavarriaga, R., Satyawan, Y. P., Schilling, F.-P., Schwenker, F., and Stadelmann, T. (2021). Two to trust: Automl for safe modelling and interpretable deep learning for robustness. In *Trustworthy AI-Integrating Learning, Optimization and Reasoning: First International Workshop, TAILOR 2020, Virtual Event, September 4–5, 2020, Revised Selected Papers 1*, pages 268–275. Springer. DOI: 10.1007/978-3-030-73959-$1_2$3.

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*. DOI: 10.1016/j.inffus.2019.12.012.

Assolin, J., Canto, G., Kreutz, D., and Feitosa, E. (2024). MH-AutoML: Transparência, interpretabilidade e desempenho na detecção de malware android. In *Anais Estendidos do XXIV Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais*, pages 113–120, Porto Alegre, RS, Brasil. SBC. DOI: 10.5753/sbseg_estendido.2024.243362.

Assolin, J., Kreutz, D., Siqueira, G., Rocha, V., Miers, C., Mansilha, R., and Feitosa, E. (2022). Droidautoml: uma ferramenta de automl para o domínio de detecção de malwares android. In *Anais Estendidos do XXII Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais*, pages 135–142, Porto Alegre, RS, Brasil. SBC. DOI: 10.5753/sbseg$_estendido$.2022.227037.

Bahri, M., Salutari, F., Putina, A., and 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*. DOI: 10.1007/s41060-022-00309-0.

Baratchi, M., Wang, C., Limmer, S., van Rijn, J. N., Hoos, H., Bäck, T., and Olhofer, M. (2024). Automated machine learning: past, present and future. *Artificial intelligence review*, 57(5):122. DOI: 10.1007/s10462-024-10726-1.

Barbudo, R., Ventura, S., and Romero, J. R. (2023). Eight years of automl: categorisation, review and trends. *Knowledge and Information Systems*, 65(12):5097–5149. DOI: 10.1007/s10115-023-01935-1.

Bifarin, O. O. and 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. DOI: 10.1021/jasms.3c00403.

Bilal, M., Ali, G., Iqbal, M. W., Anwar, M., Malik, M. S. A., and Kadir, R. A. (2022). Auto-prep: Efficient and automated data preprocessing pipeline. *IEEE Access*. DOI: 10.1109/ACCESS.2022.3198662.

Bragança, H., Rocha, V., Barcellos, L., Souto, E., Kreutz, D., and Feitosa, E. (2023). Capturing the behavior of android malware with mh-100k: A novel and multidimensional dataset. In *Anais do XXIII Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais*, pages 510–515, Porto Alegre, RS, Brasil. SBC. DOI: 10.5753/sbseg.2023.233596.

Brännström, M. (2023). Transparency of complex systems: The semantic transparency framework. Available at: https://scholar.google.com/citations?view_op=view_citation&hl=sv&user=xoWqZtYAAAAJ&citation_for_view=xoWqZtYAAAAJ:3fE2CSJIrl8C.

Carvalho, D. V., Pereira, E. M., and Cardoso, J. S. (2019). Machine learning interpretability: A survey on methods and metrics. *Electronics*. DOI: 10.3390/electronics8080832.

Colaco, C. W., Bagwe, M. D., Bose, S. A., and Jain, K. (2021). DefenseDroid: A Modern Approach to Android Malware Detection. *Strad Research*, 8(5):271–282. DOI: 10.37896/sr8.5/027.

da Silva, M. C., Licari, B., Tavares, G. M., and Junior, S. B. (2024). Benchmarking automl clustering frameworks. In *AutoML Conference 2024 (ABCD Track)*. DOI: 10.3390/info15010063.

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research*, 7(Jan):1–30. Available at: https://jmlr.org/papers/v7/demsar06a.html.

Doke, A. and Gaikwad, M. (2021). Survey on automated machine learning (automl) and meta learning. In *2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, pages 1–5. IEEE. DOI: 10.1109/ICCCNT51525.2021.9579526.

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

Ferreira, L., Pilastri, A., Martins, C. M., Pires, P. M., and Cortez, P. (2021). A comparison of AutoML tools for machine learning, deep learning and xgboost. In *IJCNN*, pages 1–8. DOI: 10.1109/IJCNN52387.2021.9534091.

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., and Hutter, F. (2015). Efficient and robust automated machine learning. *Advances in neural information processing systems*, 28. DOI: 10.1007/978-3-030-05318-5_6.

Garouani, M. and Bouneffa, M. (2023). Unlocking the black box: Towards interactive explainable automated machine learning. In *International Conference on Intelligent Data Engineering and Automated Learning*, pages 458–469. Springer. DOI: 10.1007/978−3-031−48232-8_42.

Gijsbers, P., Bueno, M. L., Coors, S., LeDell, E., Poirier, S., Thomas, J., Bischl, B., and Vanschoren, J. (2024). Amlb: an automl benchmark. *Journal of Machine Learning Research*. DOI: 10.48550/arXiv.2207.12560.

Giovanelli, J., Bilalli, B., and Abelló, A. (2022). Data preprocessing pipeline generation for autoetl. *Information Systems*. DOI: 10.1016/j.is.2021.101957.

Guerra-Manzanares, A., Bahsi, H., and Nõmm, S. (2021). KronoDroid: Time-based Hybrid-featured Dataset for Effective Android Malware Detection and Characterization. *Computers & Security*, 110:102399. DOI: 10.1016/j.cose.2021.102399.

Hariri-Ardebili, M. A., Mahdavi, P., and Pourkamali-Anaraki, F. (2024). Benchmarking automl solutions for concrete strength prediction: Reliability, uncertainty, and dilemma. *Construction and Building Materials*, 423:135782. DOI: 10.1016/j.conbuildmat.2024.135782.

Hasan, R., Dattana, V., Mahmood, S., and Hussain, S. (2024). Towards transparent diabetes prediction: Combining automl and explainable ai for improved clinical insights. *Information*, 16(1):7. DOI: 10.3390/info16010007.

Hutter, F., Kotthoff, L., and Vanschoren, J. (2019). *Automated machine learning: methods, systems, challenges*. Springer Nature. DOI: 10.1007/978-3-030-05318-5.

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

Karmaker S. et. al. (2021). Automl to date and beyond: Challenges and opportunities. *ACM Computing Surveys*, 54(8). DOI: 10.1145/3470918.

Kovalevsky, V., Delhibabu, R., and Zhukova, N. (2024). Automl framework for physical activities recognition. In *2024 3rd International Conference on Artificial Intelligence For Internet of Things (AIIoT)*, pages 1–5. IEEE. DOI: 10.1109/AIIoT58432.2024.10574646.

Krzywanski, J., Sztekler, K., Skrobek, D., Grabowska, K., Ashraf, W. M., Sosnowski, M., Ishfaq, K., Nowak, W., and Mika, L. (2024). AutoML-based predictive framework for predictive analysis in adsorption cooling and desalination systems. *Energy Science & Engineering*, 12(5):1969–1986. DOI: 10.1002/ese3.1725.

Kundu, P. P., Anatharaman, L., and Truong-Huu, T. (2021). An empirical evaluation of automated machine learning techniques for malware detection. In *Proceedings of the 2021 ACM Workshop on Security and Privacy Analytics*. DOI: 10.1145/3445970.3451155.

Le, T. T., Fu, W., and Moore, J. H. (2020). Scaling tree-based automated machine learning to biomedical big data with a feature set selector. *Bioinformatics*, 36(1):250–256. DOI: 10.1093/bioinformatics/btz470.

LeDell, E. and Poirier, S. (2020). H2O AutoML: Scalable automatic machine learning. In *Proceedings of the AutoML Workshop at ICML*, volume 2020. Available at: https://icml.cc/virtual/2020/7036.

Love, P. E., Fang, W., Matthews, J., Porter, S., Luo, H., and Ding, L. (2023). Explainable artificial intelligence (xai): Precepts, models, and opportunities for research in construction. *Advanced Engineering Informatics*. DOI: 10.1016/j.aei.2023.102024.

Lundberg, S. M. and Lee, S.-I. (2017). A unified approach to interpreting model predictions. *Advances in neural information processing systems*. DOI: 10.48550/arXiv.1705.07874.

Mahindru, A. (2018). Android permission dataset. *Mendeley Data*. Available at: https://data.mendeley.com/datasets/8y543xvnsv/1.

Mahmood, S., Hasan, R., Hussain, S., and Adhikari, R. (2025). An interpretable and generalizable machine learning model for predicting asthma outcomes: Integrating automl and explainable ai techniques. *World*, 6(1):15. DOI: 10.3390/world6010015.

Martín, A., Calleja, A., Menéndez, H. D., Tapiador, J., and Camacho, D. (2016). Adroit: Android malware detection using meta-information. In *2016 IEEE Symposium Series*

on Computational Intelligence (SSCI), pages 1–8. IEEE. DOI: 10.1109/SSCI.2016.7849904.

MH-AutoML (2024). MH-AutoML. `https://github.com/SBSegSF24/MH-AutoML`.

Mi, J.-X., Li, A.-D., and Zhou, L.-F. (2020). Review study of interpretation methods for future interpretable machine learning. *IEEE Access*. DOI: 10.1109/ACCESS.2020.3032756.

Moayeri, M., Rabbat, M., Ibrahim, M., and Bouchacourt, D. (2024). Embracing diversity: Interpretable zero-shot classification beyond one vector per class. In *Proceedings of the 2024 ACM Conference on Fairness, Accountability, and Transparency*, pages 2302–2321. DOI: 10.1145/3630106.3659039.

Molino, P., Dudin, Y., and Miryala, S. S. (2019). Ludwig: a type-based declarative deep learning toolbox. *arXiv preprint arXiv:1909.07930*. DOI: 10.48550/arXiv.1909.07930.

Narkar, S., Zhang, Y., Liao, Q. V., Wang, D., and Weisz, J. D. (2021). Model lineupper: Supporting interactive model comparison at multiple levels for automl. In *Proceedings of the 26th International Conference on Intelligent User Interfaces*, pages 170–174. DOI: 10.1145/3397481.3450658.

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*. DOI: 10.1016/j.iintel.2023.100028.

Nasimian, A. et. al. (2024). Alphaml: A clear, legible, explainable, transparent, and elucidative binary classification platform for tabular data. *Patterns*, 5(1). DOI: 10.1101/2023.06.20.545752.

Neto, H. A., Alves, R. C., and Campos, S. V. (2020). Nasirt: Automl based learning with instance-level complexity information. *arXiv preprint arXiv:2008.11846*. DOI: 10.48550/arXiv.2008.11846.

Oakes, B. J., Famelis, M., and Sahraoui, H. (2024). Building domain-specific machine learning workflows: A conceptual framework for the state of the practice. *ACM Transactions on Software Engineering and Methodology*, 33(4):1–50. DOI: 10.1145/3638243.

Oliveira, S. d., Topsakal, O., and Toker, O. (2024). Benchmarking automated machine learning (automl) frameworks for object detection. *Information*, 15(1):63. DOI: 10.3390/info15010063.

Płońska, A. and Płoński, P. (2021). Mljar: State-of-the-art automated machine learning framework for tabular data. *Version 0.10*, 3. Available at: `https://github.com/mljar/mljar-supervised`.

Ribeiro, M. T., Singh, S., and Guestrin, C. (2016). " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. DOI: 10.1145/2939672.2939778.

Rocha, V., Bragança, H., Kreutz, D., and Feitosa, E. (2024). Mh-fsf: um framework para reprodução, experimentação e avaliação de métodos de seleção de características. In *Anais Estendidos do XXIV Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais*, pages 121–128. SBC. DOI: 10.5753/sbseg_estendido.2024.243363.

Salehin, I., Islam, M. S., Saha, P., Noman, S., Tuni, A., Hasan, M. M., and Baten, M. A. (2024). AutoML: A systematic review on automated machine learning with neural architecture search. *Journal of Information and Intelligence*, 2(1):52–81. DOI: 10.1016/j.jiixd.2023.10.002.

Santu, S. K. K., Hassan, M. M., Smith, M. J., Xu, L., Zhai, C., and Veeramachaneni, K. (2021). AutoML to date and beyond: Challenges and opportunities. *ACM Computing Surveys*. DOI: 10.1145/3470918.

Schaller, M., Kruse, M., Ortega, A., Lindauer, M., and Rosenhahn, B. (2025). AutoML for multi-class anomaly compensation of sensor drift. *Measurement*, 250:117097. DOI: 10.1016/j.measurement.2025.117097.

Singh, A., Patel, S., Bhadani, V., Kumar, V., and Gaurav, K. (2024). Automl-gwl: automated machine learning model for the prediction of groundwater level. *Engineering Applications of Artificial Intelligence*, 127:107405. DOI: 10.1016/j.engappai.2023.107405.

Sırt, M. and 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. DOI: 10.52876/jcs.1604889.

SISTO, A. (2012). Androcrawl: studying alternative android marketplaces. Available at: `https://www.politesi.polimi.it/handle/10589/88407`.

Tian, J. and Che, C. (2024). Automated machine learning: A survey of tools and techniques. *Journal of Industrial Engineering and Applied Science*, 2(6):71–76. DOI: 10.70393/6a69656173.323336.

Trirat, P., Jeong, W., and Hwang, S. J. (2024). Automl-agent: A multi-agent llm framework for full-pipeline automl. *arXiv preprint arXiv:2410.02958*. DOI: 10.48550/arXiv.2410.02958.

Truong, A., Walters, A., Goodsitt, J., Hines, K., Bruss, C. B., and Farivar, R. (2019a). Towards automated machine learning: Evaluation and comparison of automl approaches and tools. *arXiv preprint arXiv:1908.05557*. DOI: 10.1109/ICTAI.2019.00209.

Truong, A., Walters, A., Goodsitt, J., Hines, K., Bruss, C. B., and Farivar, R. (2019b). Towards automated machine learning: Evaluation and comparison of AutoML approaches and tools. In *IEEE 31st ICTAI*. DOI: 10.1109/ICTAI.2019.00209.

Vakhrushev, A., Ryzhkov, A., Savchenko, M., Simakov, D., Damdinov, R., and Tuzhilin, A. (2021). Lightautoml: Automl solution for a large financial services ecosystem. *arXiv preprint arXiv:2109.01528*. DOI: 10.48550/arXiv.2109.01528 .

Vilone, G. and Longo, L. (2020). Explainable artificial intelligence: a systematic review. *arXiv preprint arXiv:2006.00093*. DOI: 10.48550/arXiv.2006.00093 .

Weerts, H., Pfisterer, F., Feurer, M., Eggensperger, K., Bergman, E., Awad, N., Vanschoren, J., Pechenizkiy, M., Bischl, B., and Hutter, F. (2023). Can fairness be automated? guidelines and opportunities for fairness-aware AutoML. *Journal of Artificial Intelligence Research*. DOI: 10.1613/jair.1.14747.

Wever, M., Tornede, A., Mohr, F., and Hüllermeier, E. (2021). Automl for multi-label classification: Overview and empirical evaluation. *IEEE transactions on pattern analysis and machine intelligence*, 43(9):3037–3054. DOI: 10.1109/TPAMI.2021.3051276.

Wu, J., Chen, X.-Y., Zhang, H., Xiong, L.-D., Lei, H., and Deng, S.-H. (2019). Hyperparameter optimization for machine learning models based on bayesian optimization. *Journal of Electronic Science and Technology*. DOI: 10.11989/JEST.1674-862X.80904120.

Wu, J., Wang, H., Ni, C., Zhang, C., and Lu, W. (2024). Data pipeline training: Integrating automl to optimize the data flow of machine learning models. In *2024 7th International Conference on Advanced Algorithms and Control Engineering (ICAACE)*, pages 730–734. IEEE. DOI: 10.1109/ICAACE61206.2024.10549260.

Yang, L. and Shami, A. (2020). On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing*, 415:295–316. DOI: 10.1016/j.neucom.2020.07.061.

Ye, T., Meng, J., Xiao, Y., Lu, Y., Zheng, A., and 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. DOI: 10.1016/j.engeos.2024.100365.

Yerima, S. Y. and Sezer, S. (2018). Droidfusion: A novel multilevel classifier fusion approach for android malware detection. *IEEE transactions on cybernetics*, 49(2):453–466. DOI: 10.1109/TCYB.2017.2777960.

Yuan, H., Yu, K., Xie, F., Liu, M., and Sun, S. (2024). Automated machine learning with interpretation: a systematic review of methodologies and applications in healthcare. *Medicine Advances*, 2(3):205–237. DOI: 10.1002/med4.75.

Zhang, X., Zhai, J., Ma, S., Guan, X., and Shen, C. (2025). Dream: Debugging and repairing automl pipelines. *ACM Transactions on Software Engineering and Methodology*, 34(4):1–29. DOI: 10.1145/3702992.

Zheng, R., Qu, L., Cui, B., Shi, Y., and Yin, H. (2023). Automl for deep recommender systems: A survey. *ACM Transactions on Information Systems*, 41(4):1–38. DOI: 10.1145/3579355.

Zimmer, L., Lindauer, M., and Hutter, F. (2020). Auto-pytorch tabular: Multi-fidelity metalearning for efficient and robust autodll. arxiv 2020. *arXiv preprint arXiv:2006.13799*. DOI: 10.48550/arXiv.2006.13799.

Zimmer, L., Lindauer, M., and Hutter, F. (2021). Auto-pytorch: multi-fidelity metalearning for efficient and robust autodl. *IEEE Trans. on Pattern Analysis and MI*, 43(9):3079–3090. DOI: 10.1109/TPAMI.2021.3067763.

Zöller, M.-A. and Huber, M. F. (2021). Benchmark and survey of automated machine learning frameworks. *Journal of artificial intelligence research*, 70:409–472. DOI: 10.48550/arXiv.1904.12054 .

Zöller, M.-A., Titov, W., Schlegel, T., and 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. DOI: 10.1145/3625240.

A Appendix - Additional Experimental Details

This section presents detailed experimental results from the comparison between MH-AutoML and seven AutoML frameworks (Auto-sklearn, AutoGluon, HyperGBM, LightAutoML, MLJAR, TPOT, and AutoPyTorch) on nine Android malware detection datasets. Section A.1 reports the transparency and interpretability evaluation. Sections A.2 and A.3 present the per-dataset results for the Normal and Balanced scenarios, respectively. Section A.4 reports computational efficiency. Section A.5 examines result variability. Section A.6 consolidates the Friedman and Holm statistical analysis. Section A.7 discusses the relationship between performance and efficiency.

The performance experiments follow a repeated evaluation protocol: each framework was run 10 times with different random seeds ($Seed_0$–$Seed_9$) on both dataset versions, yielding 90 observations per framework per version. Per-dataset results are reported as median $\pm$ standard deviation across the 10 seeds. The transparency evaluation in Section A.1 is based on a single structured assessment per framework and does not involve repeated runs.

A.1 Transparency and Interpretability Evaluation

Each framework was assessed across five categories: Functional Description, Statistical Analysis, Algorithmic Transparency, Interpretability, and Internal Analysis. The scoring criteria and full rubric are described in Section **??**.

In Functional Description, all eight frameworks received full marks (100 points), reflecting adequate documentation of their general workflows. In Statistical Analysis, MH-AutoML and MLJAR led with 100 points; AutoGluon and HyperGBM scored 80 points; the remaining four frameworks (LightAutoML, Auto-sklearn, TPOT, and AutoPyTorch) scored between 60 and 70 points, with deductions primarily related to incomplete reporting of data history and feature importance.

Algorithmic Transparency showed the widest spread. MH-AutoML was the only framework to receive full marks (100 points). AutoGluon, AutoPyTorch, LightAutoML, HyperGBM, and MLJAR each scored 75 points, while Auto-sklearn and TPOT scored 50 points. The main sources of deduction were insufficient documentation of class balancing procedures and absence of system logs.

In the Interpretability category, MH-AutoML and MLJAR led with 58.33 points. HyperGBM scored 50 points; LightAutoML and AutoGluon scored 41.67 points; AutoPyTorch scored 33.33 points; Auto-sklearn scored 25 points; and TPOT scored 8.33 points. The gaps reflect differences in support for intrinsic interpretability methods, domain-specific explanations, and model-agnostic techniques such as SHAP and LIME.

In Internal Analysis, TPOT, HyperGBM, and MLJAR each scored 100 points, performing well in model structure visualization. AutoGluon, MH-AutoML, and Auto-sklearn scored 50 points. AutoPyTorch and LightAutoML scored 0 points in this category.

Figure 18 summarizes these results across all five categories.

A.2 Results of the Normal Version

The Normal version preserves the original class distribution of each *dataset*. This scenario evaluates the performance of the *frameworks* under imbalanced conditions that reflect the natural prevalence of *malware* and benign samples.

A.2.1 Precision

Table 8 presents the precision results per *dataset* for the Normal version. MH-AutoML recorded the highest median precision in Androcrawl (0.9852) and Mh100 1K Combined (0.9708). In Androcrawl, the difference to the second-best framework (MLJAR, 0.9726) was 1.26 percentage points; in Mh100 1K Combined, the difference to the second-best (HyperGBM, 0.9291) was 4.17 percentage points.

For the remaining *datasets*, LightAutoML and MLJAR ranked among the top positions. AutoPyTorch recorded the best precision in four *datasets*: Android Permissions (0.6698), Defensedroid Apicalls Closeness (0.9661), Defensedroid Prs (0.9452), and Drebin215 (0.9915). HyperGBM showed the widest spread across *datasets*, with standard deviations up to 0.0997 in Android Permissions and 0.0284 in Kronodroid Emulator.

A.2.2 Recall

Table 9 presents the *recall* results for the Normal version. MH-AutoML recorded the highest median *recall* in three *datasets*: Adroit (0.9057), Androcrawl (0.9853), and Mh100 1K Combined (0.9712). In Adroit, the difference to the second-best framework (MLJAR, 0.8757) was 3.00 percentage points. In Android Permissions, AutoPyTorch recorded the highest *recall* (0.9981), though the corresponding MCC for this *dataset* is near zero (0.0098), as discussed in Section A.2.3.

LightAutoML recorded the best *recall* in four *datasets* (Defensedroid Apicalls Closeness, Drebin215, Kronodroid Emulator, and Kronodroid Real), with values above 0.95 and standard deviations ≤ 0.0025 in all four. HyperGBM recorded the lowest *recall* in most *datasets*, with a value of 0.8428 in Adroit and a standard deviation of 0.0285 in Kronodroid Emulator.

A.2.3 Matthews Correlation Coefficient

Table 10 presents the MCC results for the Normal version. MCC considers all entries of the confusion matrix and is therefore less susceptible to class imbalance effects than precision or recall individually. MH-AutoML recorded the best MCC in Adroit (0.7697). AutoGluon led in Defensedroid Apicalls Closeness (0.9003), AutoPyTorch in Kronodroid Emulator (0.9423) and Defensedroid Prs (0.8461), and TPOT in Kronodroid Real (0.9454) and Mh100 1K Combined (0.9044).

Android Permissions returned low MCC values across all *frameworks*, with results below 0.22. AutoPyTorch recorded a value of 0.0098 in this *dataset*, which is consistent with the high recall (0.9981) and low precision (0.6698) observed in the same *dataset*: the pattern suggests that the model predicted the majority class for most samples. HyperGBM showed the widest spread of MCC values, with standard deviations of 0.0726 in Drebin215 and 0.0754 in Kronodroid Emulator.

A.3 Results of the Balanced Version

The Balanced version applies random *undersampling* to equalize the class distribution before the train/test split. This scenario evaluates how the *frameworks* perform when the training data is balanced and how this affects the precision–*recall* relationship.

A.3.1 Precision

Table 11 presents the precision results for the Balanced version. MH-AutoML recorded the highest median precision in three *datasets*: Adroit (0.8949), Androcrawl (0.9770), and Mh100 1K Combined (0.9633). In Androcrawl, the difference to the second-best framework (Auto-sklearn, 0.9346) was 4.24 percentage points. In Mh100 1K Combined, the difference to the second-best framework (TPOT, 0.8489) was 11.44 percentage points, the largest margin across all dataset-level comparisons in this study.

AutoPyTorch recorded a median precision of 0.2159 in Adroit under balanced conditions. This value is consistent with the recall observed for the same *dataset* and version (0.2257) and with the MCC (0.1137), reported in Section A.3.3. In Drebin215, AutoPyTorch recorded precision of 0.9783 while recall fell to 0.8460, a pattern not observed in the other *frameworks* for this *dataset*, all of which recorded recall above 0.95.

A.3.2 Recall

Table 12 presents the *recall* results for the Balanced version. MH-AutoML again recorded the highest median *recall* in Adroit (0.8957) and Androcrawl (0.9741). In Mh100 1K Combined, AutoPyTorch recorded the best *recall* (0.9635), 1.08 percentage points above MH-AutoML (0.9527). In this *dataset* and version, precision and MCC for AutoPyTorch are within the range of other *frameworks*, which differs from the pattern observed in Adroit. MLJAR recorded the best *recall* in four *datasets* (Android Permissions, Defensedroid Prs, Drebin215, and Kronodroid Emulator).

In Androcrawl, the difference between the lowest *recall* (HyperGBM, 0.9684) and the highest (MH-AutoML, 0.9741) was 0.57 percentage points under balanced conditions. AutoPyTorch recorded a *recall* of 0.2257 in Adroit, consistent with the precision result reported in Section A.3.1. In Drebin215, its *recall* of 0.8460 was below all other *frameworks*, which ranged from 0.9579 to 0.9880.

A.3.3 Matthews Correlation Coefficient

Table 13 presents the MCC results for the Balanced version. Most *frameworks* recorded higher MCC values in the Balanced version compared to the Normal version. MH-AutoML showed a smaller increase relative to the other *frameworks*, which widened the gap to the leading *frameworks* in this version. MLJAR recorded the best MCC in five of nine *datasets*, LightAutoML in three, and TPOT in one.

Table 8. Precision per *dataset* – Normal version. Values are median (± standard deviation) across 10 *seeds*. **Bold** indicates the best result per *dataset*.

Dataset	MH-AutoML	Auto-sklearn	AutoGluon	HyperGBM	LightAutoML	MLJAR	TPOT	AutoPyTorch
ADR	,9067 (,0051)	,9160 (,0047)	,9094 (,0085)	,9146 (,0035)	**,9168 (,0052)**	,9153 (,0045)	,9130 (,0068)	,8810 (,0065)
ACR	**,9852 (,0012)**	,9686 (,0027)	,9698 (,0040)	,9674 (,0038)	,9701 (,0021)	,9726 (,0018)	,9696 (,0022)	,9426 (,0101)
APM	,6537 (,0074)	,6427 (,0129)	,6338 (,0166)	,6466 (,0997)	,6465 (,0065)	,6529 (,0113)	,6383 (,0143)	**,6698 (,0036)**
DAC	,9351 (,0067)	,9485 (,0007)	,9503 (,0047)	,9302 (,0156)	,9527 (,0025)	,9506 (,0030)	,9412 (,0022)	**,9661 (,0086)**
DPR	,9013 (,0083)	,9284 (,0030)	,9271 (,0062)	,9070 (,0204)	,9268 (,0034)	,9275 (,0034)	,9184 (,0065)	**,9452 (,0095)**
D215	,9676 (,0061)	,9883 (,0016)	,9883 (,0028)	,9815 (,0277)	,9902 (,0021)	,9882 (,0022)	,9876 (,0013)	**,9915 (,0085)**
K-Emu	,9473 (,0036)	,9717 (,0023)	,9732 (,0015)	,9516 (,0284)	,9737 (,0011)	**,9745 (,0016)**	,9668 (,0041)	,9688 (,0060)
K-Real	,9550 (,0042)	,9762 (,0009)	,9769 (,0013)	,9632 (,0276)	**,9780 (,0011)**	,9777 (,0013)	,9715 (,0029)	,9765 (,0101)
MH100	**,9708 (,0007)**	,9273 (,0025)	,9212 (,0079)	,9291 (,0033)	,9243 (,0042)	,9183 (,0045)	,9247 (,0041)	,8678 (,0068)

Note: ADR=Adroit, ACR=Androcrawl, APM=Android Permissions, DAC=Defensedroid Apicalls Closeness, DPR=Defensedroid Prs, D215=Drebin215, K-Emu=Kronodroid Emulator, K-Real=Kronodroid Real, MH100=Mh100 1K Combined.

Table 9. *Recall* per *dataset* – Normal version. Values are median (± standard deviation) across 10 *seeds*. **Bold** indicates the best result per *dataset*.

Dataset	MH-AutoML	Auto-sklearn	AutoGluon	HyperGBM	LightAutoML	MLJAR	TPOT	AutoPyTorch
ADR	**,9057 (,0045)**	,8763 (,0032)	,8729 (,0058)	,8428 (,0145)	,8699 (,0040)	,8757 (,0052)	,8732 (,0030)	,7959 (,0066)
ACR	**,9853 (,0012)**	,9625 (,0029)	,9575 (,0049)	,9522 (,0084)	,9605 (,0029)	,9589 (,0035)	,9595 (,0036)	,9207 (,0046)
APM	,6709 (,0024)	,5430 (,0026)	,5526 (,0069)	,5170 (,0119)	,5435 (,0032)	,5415 (,0039)	,5418 (,0056)	**,9981 (,0039)**
DAC	,9349 (,0068)	,9476 (,0005)	,9500 (,0048)	,9299 (,0157)	**,9526 (,0025)**	,9500 (,0031)	,9426 (,0019)	,9304 (,0068)
DPR	,9010 (,0083)	,9266 (,0030)	**,9269 (,0061)**	,9065 (,0204)	,9263 (,0035)	,9271 (,0034)	,9166 (,0065)	,8973 (,0098)
D215	,9676 (,0060)	,9879 (,0024)	,9872 (,0026)	,9789 (,0273)	**,9885 (,0018)**	,9877 (,0024)	,9866 (,0019)	,9738 (,0073)
K-Emu	,9470 (,0038)	,9724 (,0022)	,9726 (,0013)	,9514 (,0285)	**,9731 (,0009)**	,9728 (,0009)	,9668 (,0042)	,9657 (,0079)
K-Real	,9549 (,0042)	,9766 (,0009)	,9781 (,0014)	,9638 (,0281)	**,9786 (,0012)**	,9786 (,0013)	,9717 (,0029)	,9675 (,0037)
MH100	**,9712 (,0007)**	,9094 (,0030)	,9158 (,0087)	,9058 (,0175)	,9164 (,0038)	,9315 (,0070)	,9166 (,0043)	,8967 (,0091)

Note: The high recall of AutoPyTorch in Android Permissions (0.9981) corresponds to a near-zero MCC (0.0098), which suggests majority-class prediction rather than discriminative classification.

Table 10. MCC per *dataset* – Normal version. Values are median (± standard deviation) across 10 *seeds*. **Bold** indicates the best result per *dataset*.

Dataset	MH-AutoML	Auto-sklearn	AutoGluon	HyperGBM	LightAutoML	MLJAR	TPOT	AutoPyTorch
ADR	**,7697 (,0115)**	,7411 (,0027)	,7315 (,0106)	,7005 (,0196)	,7316 (,0084)	,7414 (,0061)	,7357 (,0074)	,7665 (,0093)
ACR	,9214 (,0062)	,9404 (,0016)	,9373 (,0050)	,9320 (,0098)	,9394 (,0034)	**,9418 (,0034)**	,9383 (,0038)	,9222 (,0103)
APM	,1190 (,0177)	,2107 (,0050)	,2060 (,0074)	,1978 (,0214)	,2108 (,0039)	**,2130 (,0033)**	,2071 (,0048)	,0098 (,0001)
DAC	,8700 (,0135)	,8492 (,0023)	**,9003 (,0094)**	,7996 (,0417)	,8622 (,0071)	,8551 (,0086)	,8301 (,0049)	,8968 (,0065)
DPR	,8023 (,0165)	,7858 (,0093)	,7915 (,0163)	,7372 (,0520)	,7900 (,0093)	,7921 (,0092)	,7642 (,0172)	**,8461 (,0027)**
D215	,9302 (,0130)	**,9702 (,0056)**	,9640 (,0073)	,9450 (,0726)	,9690 (,0053)	,9645 (,0058)	,9630 (,0043)	,9681 (,0125)
K-Emu	,8933 (,0075)	,9179 (,0060)	,9207 (,0040)	,8601 (,0754)	,9221 (,0029)	,9227 (,0035)	,9033 (,0117)	**,9423 (,0123)**
K-Real	,9095 (,0084)	,9300 (,0020)	,9344 (,0137)	,8934 (,0750)	,9358 (,0034)	,9354 (,0038)	**,9454 (,0056)**	,9419 (,0078)
MH100	,8320 (,0041)	,8693 (,0033)	,8680 (,0056)	,8639 (,0130)	,8709 (,0033)	,8763 (,0049)	**,9044 (,0037)**	,8666 (,0056)

Note: MCC ranges from -1 (total disagreement) to $+1$ (perfect classification). The low MCC values in Android Permissions are common to all frameworks. AutoPyTorch's value of 0.0098 in this dataset is accompanied by recall of 0.9981 and precision of 0.6698, consistent with a majority-class prediction pattern.

Table 11. Precision per *dataset* – Balanced version. Values are median (± standard deviation) across 10 *seeds*. **Bold** indicates the best result per *dataset*.

Dataset	MH-AutoML	Auto-sklearn	AutoGluon	HyperGBM	LightAutoML	MLJAR	TPOT	AutoPyTorch
ADR	**,8949 (,0051)**	,8751 (,0091)	,8754 (,0106)	,8873 (,0127)	,8859 (,0072)	,8838 (,0097)	,8792 (,0120)	,2159 (,0018)
ACR	**,9770 (,0025)**	,9346 (,0021)	,9297 (,0058)	,9302 (,0103)	,9292 (,0040)	,9277 (,0040)	,9314 (,0057)	,7523 (,0047)
APM	,6530 (,0077)	,6087 (,0069)	,6074 (,0053)	,5989 (,0074)	,6069 (,0064)	,6143 (,0048)	,6124 (,0064)	**,6763 (,0066)**
DAC	,9335 (,0035)	,9491 (,0033)	,9511 (,0024)	,9294 (,0169)	**,9537 (,0025)**	,9487 (,0043)	,9446 (,0050)	,9445 (,0070)
DPR	,8952 (,0051)	,9259 (,0016)	,9246 (,0042)	,9055 (,0218)	,9266 (,0036)	**,9292 (,0048)**	,9178 (,0047)	,9241 (,0058)
D215	,9580 (,0068)	,9843 (,0004)	,9859 (,0021)	,9776 (,0250)	,9877 (,0020)	**,9898 (,0013)**	,9871 (,0027)	,9783 (,0064)
K-Emu	,9401 (,0039)	,9711 (,0024)	,9723 (,0022)	,9506 (,0255)	,9718 (,0010)	**,9734 (,0019)**	,9651 (,0026)	,9273 (,0065)
K-Real	,9503 (,0018)	,9766 (,0001)	,9766 (,0012)	,9633 (,0281)	**,9776 (,0010)**	,9769 (,0012)	,9722 (,0022)	,9628 (,0074)
MH100	**,9633 (,0028)**	,8455 (,0016)	,8371 (,0073)	,8241 (,0135)	,8391 (,0073)	,8325 (,0096)	,8489 (,0072)	,6857 (,0035)

Note: AutoPyTorch's precision in Adroit (0.2159) is accompanied by recall of 0.2257 and MCC of 0.1137, indicating that the model did not converge under balanced conditions for this dataset.

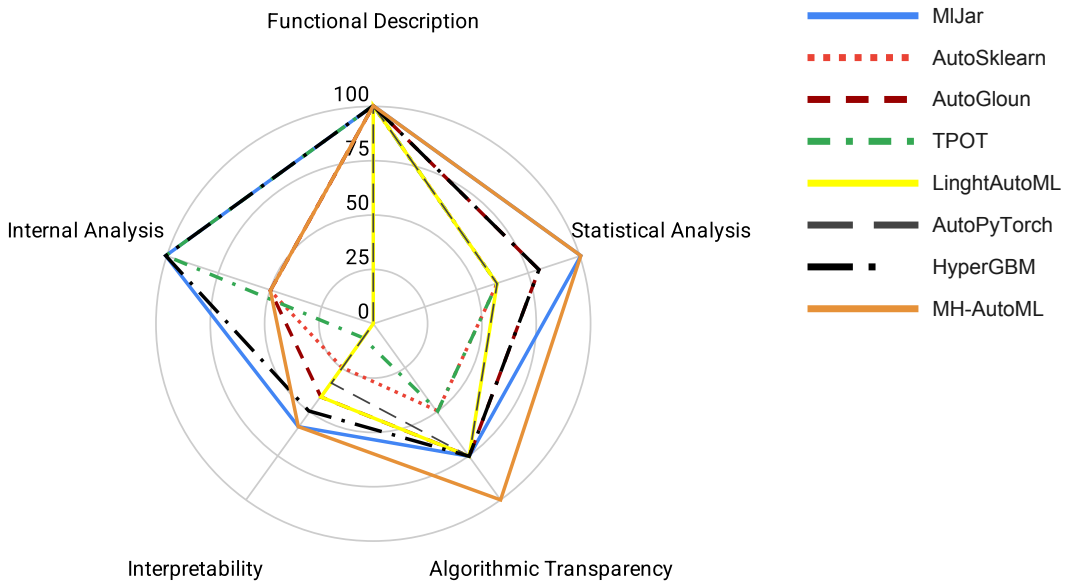

Figure 18. Scores per framework across the five transparency and interpretability categories.

Table 12. *Recall* per *dataset* – Balanced version. Values are median (± standard deviation) across 10 *seeds*. **Bold** indicates the best result per *dataset*.

Dataset	MH-AutoML	Auto-sklearn	AutoGluon	HyperGBM	LightAutoML	MLJAR	TPOT	AutoPyTorch
ADR	**,8957 (,0055)**	,8798 (,0040)	,8780 (,0058)	,8656 (,0138)	,8792 (,0067)	,8836 (,0057)	,8787 (,0071)	,2257 (,0026)
ACR	**,9741 (,0035)**	,9729 (,0004)	,9723 (,0014)	,9684 (,0103)	,9728 (,0014)	,9734 (,0014)	,9731 (,0015)	,9627 (,0079)
APM	,6076 (,0103)	,6135 (,0059)	,6195 (,0060)	,6092 (,0083)	,6190 (,0071)	**,6275 (,0056)**	,6255 (,0063)	,5343 (,0037)
DAC	,9330 (,0037)	,9488 (,0026)	,9509 (,0024)	,9292 (,0169)	**,9535 (,0025)**	,9485 (,0048)	,9465 (,0050)	,8804 (,0068)
DPR	,8944 (,0051)	,9247 (,0012)	,9242 (,0041)	,9050 (,0217)	,9257 (,0036)	**,9288 (,0047)**	,9175 (,0034)	,9182 (,0067)
D215	,9579 (,0068)	,9852 (,0000)	,9868 (,0021)	,9789 (,0229)	,9878 (,0019)	**,9880 (,0012)**	,9863 (,0031)	,8460 (,0089)
K-Emu	,9396 (,0039)	,9712 (,0018)	,9726 (,0021)	,9511 (,0250)	,9722 (,0009)	**,9728 (,0013)**	,9661 (,0034)	,9078 (,0072)
K-Real	,9502 (,0018)	,9772 (,0000)	,9773 (,0012)	,9643 (,0287)	**,9783 (,0010)**	,9774 (,0012)	,9731 (,0019)	,9195 (,0067)
MH100	,9527 (,0063)	,9470 (,0009)	,9517 (,0026)	,9461 (,0126)	,9514 (,0026)	,9524 (,0029)	,9506 (,0067)	**,9635 (,0085)**

The MCC of MH-AutoML on balanced data ranged from 0.2129 (Android Permissions) to 0.9098 (Drebin215). The largest difference to the best-performing *framework* was in Androcrawl, where MH-AutoML (0.8743) was 6.73 percentage points below Auto-sklearn (0.9416). AutoPyTorch recorded an MCC of 0.1137 in Adroit, the lowest value in the Balanced version outside of Android Permissions. In Drebin215, AutoPyTorch's MCC of 0.8655 was below the range of the other *frameworks* (0.9098 to 0.9789), which is consistent with its recall of 0.8460 in the same *dataset*. HyperGBM recorded standard deviations of 0.0564 in Kronodroid Real and 0.0501 in Kronodroid Emulator, maintaining the variability pattern observed in the Normal version.

A.4 Computational Efficiency

Table 14 presents the average execution times for all *frameworks* across both versions. HyperGBM recorded the shortest times in both versions, with averages of 45 seconds (Normal) and 47 seconds (Balanced). MH-AutoML ranked third, after HyperGBM and AutoGluon, with averages of 10 minutes and 35 seconds (Normal) and 6 minutes and 36 seconds (Balanced). AutoPyTorch recorded the longest times, averaging above 2 hours on the Normal data and 1 hour and 42 minutes on the Balanced data.

The Friedman test indicated significant differences in execution time (Normal: $\chi^2 = 582.72$, $p < 0.001$; Balanced: $\chi^2 = 597.67$, $p < 0.001$). The Friedman average ranking placed HyperGBM first (1.11 Normal; 1.00 Balanced), followed by AutoGluon (2.42; 2.51) and MH-AutoML (2.68; 2.56). Holm *post-hoc* tests confirmed that MH-AutoML ran faster than Auto-sklearn, LightAutoML, MLJAR, TPOT, and AutoPyTorch (all $p < 0.001$) in both versions, and slower than HyperGBM ($p < 0.001$) and AutoGluon ($p < 0.05$). Based on the mean times in Table 14, MH-AutoML completed its *pipeline* approximately $5.8\times$ faster than Auto-sklearn, $6.1\times$ faster than MLJAR, $5.7\times$ faster than TPOT, and $11.7\times$ faster than AutoPyTorch on the Normal data.

A.5 Variability Analysis

To evaluate the consistency of each *framework* across different random *seeds*, we examine the distribution of results using *boxplots* and the coefficient of variation (CV). Figure 19 presents global *boxplots* aggregating all 90 observations (9 datasets $\times$ 10 seeds) per *framework* for each metric and version.

In Figure 19, MH-AutoML shows a narrow distribution for precision in both versions, with few *outliers*. AutoPyTorch shows the widest spread in the Balanced data, largely driven by its results in Adroit (precision 0.2159, recall 0.2257), which appear as low-end *outliers*. HyperGBM consistently shows the largest interquartile range among the *frameworks* across all metrics.

For MCC, differences between *frameworks* are more pronounced than for precision or recall. In the Balanced data, the

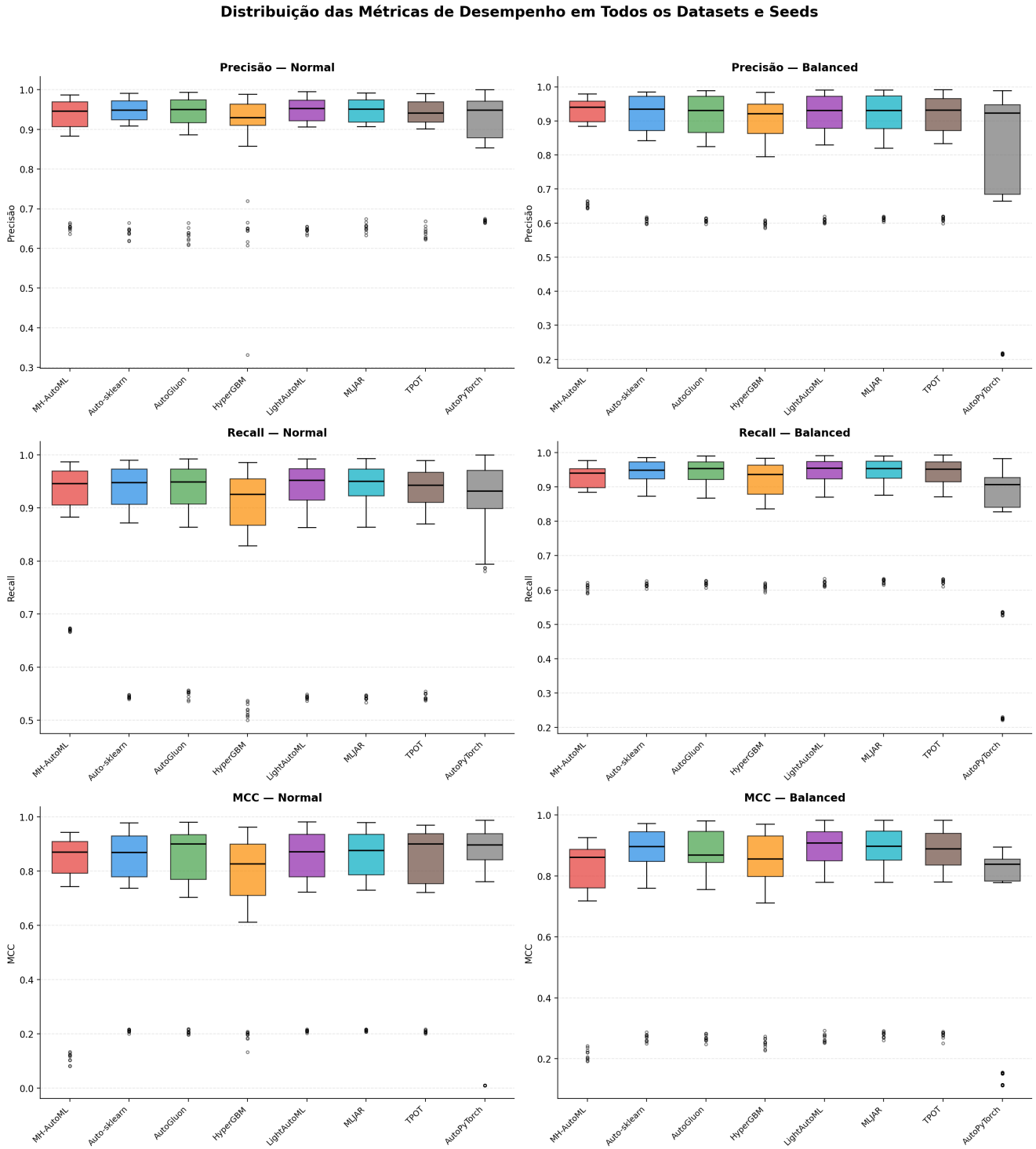

Figure 19. Global distribution of precision, *recall*, and MCC across all *datasets* and *seeds* for each *framework*. Boxes represent the IQR; *whiskers* extend to 1.5×IQR; circles indicate *outliers*.

Table 13. MCC per *dataset* – Balanced version. Values are median (± standard deviation) across 10 *seeds*. **Bold** indicates the best result per *dataset*.

Dataset	MH-AutoML	Auto-sklearn	AutoGluon	HyperGBM	LightAutoML	MLJAR	TPOT	AutoPyTorch
ADR	,7484 (,0118)	,7880 (,0143)	,7872 (,0147)	,7960 (,0152)	,7988 (,0111)	**,8004 (,0137)**	,7983 (,0140)	,1137 (,0006)
ACR	,8743 (,0143)	**,9416 (,0015)**	,9388 (,0044)	,9368 (,0043)	,9386 (,0028)	,9382 (,0022)	,9403 (,0023)	,8384 (,0094)
APM	,2129 (,0169)	,2710 (,0112)	,2671 (,0104)	,2523 (,0142)	,2663 (,0128)	,2804 (,0093)	**,2824 (,0109)**	,1525 (,0014)
DAC	,8665 (,0072)	,8960 (,0069)	,8575 (,0068)	,8593 (,0340)	**,9074 (,0052)**	,8969 (,0093)	,8878 (,0081)	,8264 (,0051)
DPR	,7895 (,0102)	,8498 (,0036)	,8489 (,0082)	,8109 (,0436)	,8526 (,0071)	**,8580 (,0097)**	,8390 (,0069)	,8473 (,0090)
D215	,9098 (,0146)	,9715 (,0002)	,9736 (,0037)	,9592 (,0452)	,9769 (,0034)	**,9789 (,0021)**	,9771 (,0041)	,8655 (,0093)
K-Emu	,8786 (,0078)	,9421 (,0035)	,9454 (,0043)	,9026 (,0501)	,9445 (,0019)	**,9466 (,0031)**	,9318 (,0052)	,8516 (,0074)
K-Real	,9002 (,0037)	,9539 (,0000)	,9541 (,0023)	,9278 (,0564)	**,9560 (,0020)**	,9544 (,0023)	,9468 (,0045)	,8807 (,0102)
MH100	,7772 (,0200)	,8692 (,0020)	,8654 (,0043)	,8508 (,0152)	,8662 (,0049)	,8628 (,0071)	**,8758 (,0041)**	,7900 (,0085)

Note: AutoPyTorch's MCC in Adroit (0.1137) is accompanied by precision of 0.2159 and recall of 0.2257. In Drebin215, its MCC of 0.8655 corresponds to recall of 0.8460, while precision remained at 0.9783.

Table 14. Average execution time per *framework* (minutes:seconds). Values are means across 9 *datasets* × 10 *seeds* per version.

Framework	Normal	Balanced
HyperGBM	0:45	0:47
AutoGluon	6:19	4:39
MH-AutoML	10:35	6:36
LightAutoML	60:41	59:12
TPOT	60:21	57:54
Auto-sklearn	61:33	56:46
MLJAR	64:38	63:01
AutoPyTorch	123:47	102:18

Note: Times are approximate means. Exact per-dataset times are available in the replication package.

median MCC of MH-AutoML is below the values of LightAutoML, MLJAR, Auto-sklearn, and TPOT. This is consistent with its rank of 6.96 in the Balanced MCC Friedman ranking (Table 15). The MCC IQR of MH-AutoML is narrower than that of HyperGBM and AutoPyTorch.

Figure 20 presents the *heatmap* of the coefficient of variation (CV), which normalizes variability relative to the mean, allowing comparison across *datasets* with different performance levels.

HyperGBM shows the highest CV values across most *datasets* and metrics. In Android Permissions, its precision CV reached 16.1% (Normal) and 1.2% (Balanced), and its MCC CV reached 11.2% (Normal) and 5.7% (Balanced). MH-AutoML shows CV values below 1.0% for precision and *recall* in most *datasets*. The exception is MCC in Android Permissions (15.9% Normal; 7.9% Balanced), a pattern shared across all *frameworks* on this *dataset*.

LightAutoML and MLJAR show CV values below 0.5% for precision and *recall* in most *datasets*. Auto-sklearn reached CV $= 0.0\%$ in Kronodroid Real Balanced for both precision and *recall*, indicating identical results across all 10 seeds.

A.6 Statistical Analysis

Table 15 consolidates the Friedman test results and global average rankings for all metrics and versions. All tests returned $p < 0.001$ with $N = 90$ blocks.

LightAutoML and MLJAR hold the top two positions across all predictive metrics, alternating between first and second place. MH-AutoML ranks 6th in precision Balanced (rank 4.71), 5th in recall Normal (rank 4.61), 7th in MCC in both versions (ranks 5.80 and 6.96), and 3rd in execution time in both versions (ranks 2.68 and 2.56).

Holm *post-hoc* pairwise comparisons between MH-AutoML and each other *framework* yielded the following results. For precision in the Normal version, LightAutoML ($p = 0.012$) and MLJAR ($p = 0.007$) returned higher values than MH-AutoML with statistical significance; no other framework did. In the Balanced version, MH-AutoML returned higher precision than HyperGBM ($p = 0.001$); no significant difference was found relative to the remaining *frameworks*. For recall in the Normal version, MH-AutoML returned higher values than HyperGBM ($p < 0.001$). In the Balanced version, five *frameworks* returned higher recall than MH-AutoML with statistical significance: MLJAR, LightAutoML, AutoGluon, Auto-sklearn, and TPOT (all $p < 0.001$). For MCC, MH-AutoML returned lower values than most *frameworks* in both versions, with statistically significant differences in the majority of pairwise comparisons; the only *framework* it ranked above with statistical support was AutoPyTorch. For execution time, MH-AutoML ran faster than Auto-sklearn, LightAutoML, MLJAR, TPOT, and AutoPyTorch in both versions (all $p < 0.001$).

A.7 Trade-off Analysis

The results across datasets, metrics, and versions indicate that MH-AutoML does not follow a uniform performance pattern. For precision, it ranks first in two or three datasets per version and is statistically equivalent to most other *frameworks* in the remaining cases. For recall and MCC, its global Friedman rankings are lower, placing it in the second half of the field across most configurations, with the Balanced MCC ranking (6.96, 7th place) being the weakest result overall.

MH-AutoML shows low CV values for precision and recall across most datasets, and its results are consistent across seeds. For execution time, it ranks third in both versions, after HyperGBM and AutoGluon. Based on the mean times in Table 14, MLJAR and LightAutoML run approximately $6.1\times$ and $5.7\times$ longer than MH-AutoML, respectively, on the Normal data.

HyperGBM records the shortest execution times but the lowest recall and MCC values, and the highest variability across both versions. AutoPyTorch records the longest execu-

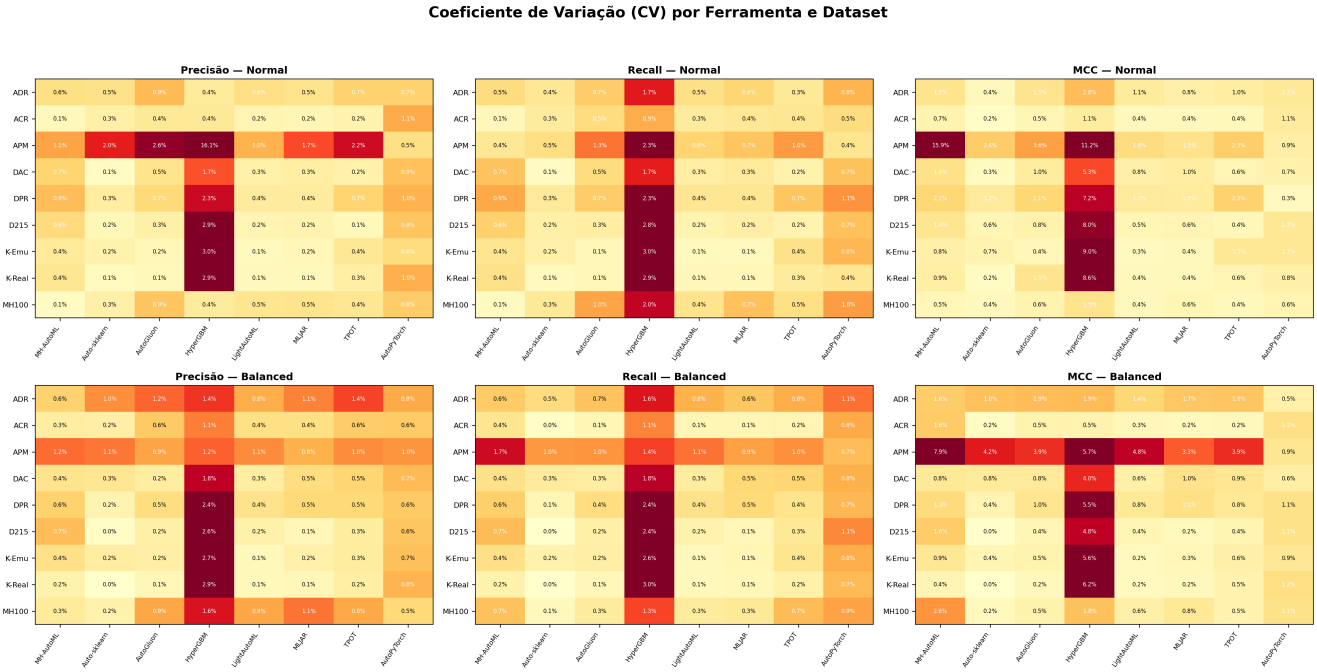

Figure 20. Coefficient of variation (CV, %) per *framework* and *dataset*. Darker cells indicate higher relative variability.

Table 15. Friedman test results and average rankings for all metrics and versions ($N = 90$ blocks per test, all $p < 0.001$).

Metric (Version)	χ^2	Average Ranking (best → worst)
Precision (Normal)	110.39	LightAutoML (3.07) > MLJAR (3.44) > Auto-sklearn (4.11) > AutoGluon (4.18) > AutoPyTorch (4.33) > TPOT (5.42) > MH-AutoML (5.52) > HyperGBM (5.92)
Precision (Balanced)	129.64	LightAutoML (3.25) > MLJAR (3.36) > Auto-sklearn (3.78) > AutoGluon (4.22) > TPOT (4.50) > MH-AutoML (4.71) > AutoPyTorch (5.96) > HyperGBM (6.23)
Recall (Normal)	222.27	LightAutoML (3.11) > MLJAR (3.16) > AutoGluon (3.46) > Auto-sklearn (3.73) > MH-AutoML (4.61) > TPOT (4.67) > AutoPyTorch (6.33) > HyperGBM (6.94)
Recall (Balanced)	269.32	MLJAR (2.44) > LightAutoML (3.13) > AutoGluon (3.52) > Auto-sklearn (3.99) > TPOT (4.08) > MH-AutoML (5.52) > HyperGBM (6.50) > AutoPyTorch (6.82)
MCC (Normal)	173.58	MLJAR (3.36) > LightAutoML (3.47) > AutoPyTorch (3.54) > AutoGluon (3.92) > Auto-sklearn (4.37) > TPOT (4.53) > MH-AutoML (5.80) > HyperGBM (7.00)
MCC (Balanced)	391.83	MLJAR (2.51) > LightAutoML (2.66) > Auto-sklearn (3.20) > TPOT (3.55) > AutoGluon (3.99) > HyperGBM (5.76) > MH-AutoML (6.96) > AutoPyTorch (7.38)
Exec. Time (Normal)	582.72	HyperGBM (1.11) > AutoGluon (2.42) > MH-AutoML (2.68) > LightAutoML (3.80) > TPOT (5.66) > Auto-sklearn (5.81) > MLJAR (6.58) > AutoPyTorch (7.94)
Exec. Time (Balanced)	597.67	HyperGBM (1.00) > AutoGluon (2.51) > MH-AutoML (2.56) > LightAutoML (3.94) > TPOT (5.61) > Auto-sklearn (5.69) > MLJAR (6.74) > AutoPyTorch (7.94)

Note: Rankings are in ascending order of rank value (lower rank = better performance). All tests are significant at $\alpha = 0.05$.

tion times and the lowest MCC in the Balanced version (rank 7.38, last place), with results in Adroit (precision 0.2159, recall 0.2257, MCC 0.1137) indicating that the model did not produce useful predictions for that dataset under balanced conditions.

AutoGluon ranks above MH-AutoML in global Friedman rankings for all predictive metrics in both versions, and it also runs faster (rank 2.42 vs. 2.68 in the Normal version). MH-AutoML records higher precision than AutoGluon in Androcrawl and Mh100 1K Combined in both versions, which are the two datasets where the difference between the two is most visible at the dataset level.