الجمـهورية الجـزائرية الديمقراطية الشعبية
REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
وزارة التعليم العالي و البحث العلمي
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
جامعة أبي بـكر بلـقايـد – تلمسان –
Université Aboubakr Belkaïd – Tlemcen –
Faculté de TECHNOLOGIE

MEMOIRE

Présenté pour l'obtention du **diplôme** de **MASTER**

En : Télécommunications

Spécialité : Réseaux et Télécommunications

Par : BENABDALLAH Fatima Zohra Chaimaa

BENYELLOUL Rania

Sujet

DETECTION D'ANOMALIES DANS LES APPLICATIONS MOBILES PAR L'INTELLIGENCE ARTIFICIELLE

Soutenu publiquement, le 15/06/2026, devant le jury composé de :

M. MOUSSAOUI Djilali	MCA	Université de Tlemcen	Président
Mme. BOUCHENAK KHELADI Yassmina	MCB	Université de Tlemcen	Examinatrice
Mme. TALEB Sarra	MCB	Université de Tlemcen	Encadrante
Mme. BENAISSA Amal	MCB	Université de Tlemcen	Co-encadrante

Année universitaire : 2025/2026

REMERCIEMENT

Ce mémoire n'aurait jamais vu le jour sans le concours de nombreuses personnes qui ont jalonné ce parcours de leur présence, de leur savoir et de leur bienveillance.

Que chacune d'entre elles trouve ici l'expression de notre sincère gratitude. Nos premiers remerciements vont naturellement à notre encadrante, qui a accepté de guider nos pas dans ce travail de recherche. Sa rigueur intellectuelle, ses remarques toujours justes et sa disponibilité, même dans les moments les plus chargés ont été pour nous un repère constant. Travailler sous sa direction a été une véritable leçon d'exigence et de méthode.

Nous remercions également les membres du jury d'avoir consacré de leur temps à la lecture et à l'évaluation de ce travail. Leurs remarques et leur regard critique ne pourront qu'enrichir notre réflexion.

Une pensée particulière va à nos familles, nos amis et collègues de promotion, surtout, qui ont porté nos rêves bien avant que nous n'y croyions nous-mêmes. Leur confiance silencieuse, leurs sacrifices discrets et leur amour sans condition ont été le socle sur lequel tout repose.

Merci pour les débats tardifs, les doutes partagés et les fous rires qui ont allégé les moments difficiles. Vous avez rendu cette aventure humaine et mémorable.

Nous adressons enfin notre reconnaissance à l'ensemble des enseignants du département Réseaux et Télécommunications, dont les enseignements ont forgé notre bagage intellectuel et notre goût pour la recherche.

Résumé

Avec la croissance rapide des smartphones et des applications mobiles, le système d'exploitation Android est devenu la plateforme mobile la plus utilisée dans le monde. Cette popularité s'accompagne d'une augmentation significative de logiciels malveillants sophistiqués, capables de contourner les mécanismes de détection traditionnels, tels que ceux basés sur les signatures et les règles comportementales statiques.

Face à cette menace, le présent mémoire propose une approche de détection d'anomalies comportementales basée sur l'apprentissage automatique non supervisé, appliquée au jeu de données de référence CICMalDroid 2020 (11 598 applications Android, 476 caractéristiques d'appels système).

Dans le cadre de cette étude, trois algorithmes complémentaires ont été évalués selon deux protocoles expérimentaux distincts : une approche one-class (entraînement exclusif sur applications bénignes) et une approche semi-supervisée avec rééquilibrage SMOTETomek. Les algorithmes étudiés sont l'Isolation Forest, l'Autoencoder et le One-Class SVM à noyau RBF.

Les résultats démontrent la supériorité de l'approche one-class, avec des F1-Scores dépassant 0,99 pour l'Isolation Forest et le One-Class SVM, sans aucune connaissance préalable des malwares lors de l'entraînement. Ces performances égalent, voire surpassent, celles de méthodes supervisées de l'état de l'art, tout en offrant la capacité de détecter des menaces inconnues (attaques zero-day). Enfin, l'interprétabilité des décisions est renforcée par l'intégration de l'analyse SHAP, renforçant ainsi la transparence et la fiabilité des systèmes de détection proposés.

Mots-clés

Détection d'anomalies, apprentissage non supervisé, malwares Android, CICMalDroid 2020, Isolation Forest, Autoencodeur, One-Class SVM, appels système (syscalls), attaques zero-day, SHAP.

Abstract

With the rapid growth of smartphones and mobile applications, the Android operating system has become the most widely used mobile platform in the world. This popularity is accompanied by a significant increase in sophisticated malware, capable of bypassing traditional detection mechanisms, such as those based on signatures and static behavioral rules.

Faced with this threat, the present thesis proposes an approach for detecting behavioral anomalies based on unsupervised machine learning, applied to the reference dataset CICMalDroid 2020 (11 598 Android applications, 476 system call features).

In this study, three complementary algorithms were evaluated according to two distinct experimental protocols: a one-class approach (training exclusively on benign applications) and a semi-supervised approach with SMOTETomek rebalancing. The algorithms studied are Isolation Forest, Autoencoder and One-Class SVM with RBF kernel.

The results demonstrate the superiority of the one-class approach, with F1-Scores exceeding 0.99 for Isolation Forest and One-Class SVM, without any prior knowledge of malware during training. These performances match or even surpass state-of-the-art supervised methods, while offering the ability to detect unknown threats (zero-day attacks).

Finally, the interpretability of decisions is enhanced by the integration of SHAP analysis, thereby strengthening the transparency and reliability of the proposed detection systems.

Keywords

Anomaly detection, unsupervised learning, Android malware, CICMalDroid 2020, Isolation Forest, Autoencoder, One-Class SVM, system calls (syscalls), zero-day attacks, SHAP

ملخص

مع النمو السريع للهواتف الذكية وتطبيقات الهاتف المحمول، أصبح نظام التشغيل أندرويد أكثر منصات الهواتف استخداما في العالم. وقد رافقت هذه الشعبية زيادة كبيرة في البرمجيات الخبيثة المتطورة، القادرة على تجاوز آليات الكشف التقليدية، مثل تلك المعتمدة على التواقيع والقواعد السلوكية الثابتة.

في مواجهة هذا التهديد، تقترح هذه الأطروحة مقاربة للكشف عن الشذوذ السلوكي تعتمد على التعلم الآلي غير الخاضع للإشراف والمطبقة على مجموعة البينات المرجعية CICMaLDroid 2020التي تضم 11598 تطبيق أندرويد و476 خاصية لاستدعاءات النظام.

في هذه الدراسة، تم تقييم ثلاث خوارزميات متكاملة وفق بروتكولين تجريبيين مختلفين: مقاربة الفئة الواحدة (التدريب حصريا على التطبيقات السليمة) ومقاربة شبه خاضعة للإشراف مع إعادة التوازن باستخدام SMOTETomek. والخوارزميات المدروسة هي: Isolation Forest و Autoencodeur و One-Class SVM باستخدام نواة RBF.

أظهرت النتائج تفوق مقاربة الفئة الواحدة، حيث تجاوزت قيم F1-Scoreنسبة 0.99 لكل من Isolation Forest و-One Class SVM، دون أي معرفة مسبقة بالبرمجيات الخبيثة أثناء التدريب. كما أن هذه النتائج تضاهي أو تتجاوز الأساليب الخاضعة للإشراف الحديثة، مع توفير القدرة على اكتشاف التهديدات غير المعروفة (هجمات zero-day)

أخيرا، تم تعزيز قابلية تفسير القرارات من خلال دمج تحليل SHAP، مما يعزز شفافية وموثوقية أنظمة الكشف المقترحة.

الكلمات المفتاحية:

كشف الحالات غير الطبيعية، التعلم غير الخاضع للإشراف، البرمجيات الخبيثة لأندرويد، Isolation Forest، CICMaLDroid 2020 , One-Class SVM , Autoencodeur , استدعاءات النظام (Syscalls)، هجمات zero-day، SHAP

Table des matières

INTRODUCTION GÉNÉRALE : 1

..... 4

Chapitre 1 : Applications mobiles et enjeux de sécurité.

1.1 INTRODUCTION : 5

1.2 FONDEMENTS DES APPLICATIONS MOBILES : 6

1.2.1 Définition et Évolution Historique : 6

1.2.1.1 Évolution historique des applications mobiles : 7

1.3 CLASSIFICATION DES APPLICATIONS MOBILES : 8

1.3.1 Applications natives : 8

1.3.2 Applications Web Progressives (PWA) : 10

1.3.3 Applications Hybrides : 11

1.4 ARCHITECTURE DE l'APPLICATION MOBILE : 11

1.4.1 Architecture Logicielle en Couches : 11

1.4.2 Architecture de la Plateforme Android : 12

1.4.2.1 Noyau Linux (Linux Kernel) : 13

1.4.2.2 Couche d'abstraction matérielle (HAL « Hardware Abstraction Layer ») : 13

1.4.2.3 Android Runtime (ART) et format DEX : 13

1.4.2.4 Application Framework et composants Android : 14

1.4.2.5 Modèle de sécurité Android : sandbox, permissions et SELinux : 14

1.4.3 Architecture de la Plateforme iOS : 15

1.4.3.1 Mécanismes de sécurité iOS avancés : 16

1.5 SECURITE DES APPLICATIONS MOBILES : 16

1.5.1 Menaces et Vulnérabilités (Référentiel OWASP Mobile) : 16

1.5.1.1 Vecteurs d'attaque spécifiques à Android : 18

1.5.2 Taxonomie des Malwares Mobiles : 19

1.5.3 Attaques courantes (exfiltration de données, comportements anormaux.) : 19

1.6 NOTION D'ANOMALIE DANS LES APPLICATIONS MOBILES : 20

1.6.1 Définition d'une anomalie : 20

1.6.2 Exemples Concrets d'Anomalies Comportementales : 20

1.7 LIMITES DES METHODES CLASSIQUES DE DETECTION : 21

1.7.1 Détection par Signatures (Antivirus Traditionnels) : 21

1.7.2 Systèmes de Détection à Règles Fixes : 21

1.7.3 Limites face aux Technique d'Attaques Modernes : 23

1.7.3.1 Polymorphisme et métamorphisme : 23

1.7.3.2 Évasion par détection de l'environnement d'analyse : 23

1.7.3.3 Menaces Persistantes Avancées mobiles (APT) : 23
1.7.3.4 Chiffrement intégral du trafic malveillant : 24
1.8 CONCLUSION 24

Chapitre 2 : Intelligence artificielle et détection d'anomalies.

2.1 INTRODUCTION : 27
2.2 LE MACHINE LEARNING (APPRENTISSAGE AUTOMATIQUE) : 28
2.2.1 Définition : 28
2.2.2 Types d'apprentissage automatique : 29
2.2.2.1 L'apprentissage supervisé : 29
2.2.2.2 L'apprentissage non supervisé : 30
2.2.2.3 L'apprentissage par renforcement : 32
2.2.2.4 Apprentissage semi supervisé : 33
2.2.3 Justification du choix de l'apprentissage non supervisé : 33
2.3 LE DEEP LEARNING (APPRENTISSAGE PROFOND) : 34
2.4 MACHINE LEARNING VS DEEP LEARNING: 34
2.5 DETECTION D'ANOMALIES PAR L'APPRENTISSAGE AUTOMATIQUE : 36
2.5.1 Principe générale : 36
2.5.2 Approches de détection d'anomalies : 37
2.5.3 Avantage de l'IA pour la détection de malwares Android : 38
2.6 ALGORITHMES DE DETECTION DANOMALIES ETUDIES : 39
2.6.1 Isolation Forest : 39
2.6.1.1 Définition : 39
2.6.1.2 L'arbre d'isolation (iTree) : 40
2.6.1.3 La zone d'isolation forestière et l'évaluation du score d'anomalie : 41
2.6.1.4 Hyperparamètres et configuration : 42
2.6.1.5 Propriétés théoriques et complexité : 42
2.6.1.6 Avantages et limites : 43
2.6.2 Autoencodeur : 44
2.6.2.1 Définition : 44
2.6.2.2 Architecture d'Autoencodeur : 44
2.6.2.3 Entrainement et fonction de perte : 46
2.6.2.4 Détection d'anomalies par erreur de reconstruction : 46
2.6.2.5 Hyperparamètres clés : 47
2.6.2.6 Avantage et limites : 47
2.6.3 One-Class SVM (OC-SVM): 48
2.6.3.1 Définition: 48
2.6.3.2 Fondements théoriques des SVM : 49

2.6.3.3 Interprétation géométrique : 49
2.6.3.4 Hyperparamètres et configuration : 51
2.6.3.5 Réglage du paramètre γ du noyau RBF : 51
2.6.3.6 Avantages et limites : 52
2.7 COMPARAISON DES TROIS ALGORITHMES : 52
2.8 CONCLUSION : 53

Chapitre 3 : Implémentation et évaluation du système de détection d'anomalies.

3.1 INTRODUCTION : 56
3.2 ENVIRONNEMENT DE DEVELOPPEMENT : 57
3.2.1 Plateforme Kaggle Notebook : 57
3.2.2 Langage et bibliothèques : 57
3.2.3 Reproductibilité et fixation du SEED : 58
3.3 PRESENTATION ET ANALYSE DU JEU DE DONNEES CICMalDroid 2020 : 59
3.3.1 Origine et Processus de Collecte : 59
3.3.2 Structure et Caractéristiques du Dataset : 60
3.3.3 Analyse Exploratoire des Données (EDA) : 62
3.3.3.1 Analyse statistique descriptive : 62
3.3.3.2 Distribution des classes et déséquilibre : 63
3.3.3.3 Analyse des corrélations : 63
3.3.3.4 Visualisation des données : 64
3.4 JUSTIFICATION DU CHOIX DES MODELES : 65
3.4.1 Isolation Forest (Détection par Isolation Stochastique) : 65
3.4.2 One-Class SVM (Modélisation Frontière à Noyau RBF) : 66
3.4.3 Autoencodeur (Représentation Latente et Reconstruction) : 67
3.5 PRETRAITEMENT DES DONNEES : 68
3.5.1 Encodage de la variable cible : 68
3.5.2 Séparation Train / Validation / Test : 68
3.5.3 Normalisation par StandardScaler : 69
3.5.4 Rééquilibrage par SMOTE (approche 2 uniquement) : 69
3.5.5 Réduction de dimensionnalité par PCA : 70
3.5.6 Métriques d'Évaluation : 72
3.6 PROTOCOLE EXPERIMENTAL : 72
3.6.1 Approche 1 (AP1) (Détection One-Class) : 72
3.6.1.1 Résultats de l'Isolation Forest (AP1) : 73
3.6.1.2 Résultats du One-Class SVM (AP1) : 74
3.6.1.3 Résultats de l'Autoencoder (AP1) : 74
3.6.2 Approche 2 (Validation croisée semi-supervisée (SMOTETomek)) : 75
3.6.2.1 Résultats de l'Isolation Forest (AP2) : 76
3.6.2.2 Résultats du One-Class SVM (AP2) : 77

3.6.2.3 Résultats de l'Autoencodeur (AP2) : 77
3.7 RESULTATS ET DISCUSSION : 78
3.7.1 Tableau comparatif global : 78
3.7.2 Comparaison Approche 1 vs Approche 2 : 80
3.7.3 Comparaison avec l'état de l'art : 82
3.8 DISCUSSION GENERALE ET LIMITES : 83
3.8.1 Forces de l'approche non supervisée : 83
3.8.2 Limites et perspectives 84
3.9 CONCLUSION : 85
CONCLUSION GENERALE : 87
Références Bibliographiques 89

LISTE DES FIGURES

Figure 1. 1 : Plateformes de distribution des applications mobiles..... 7

Figure 2. 1 : le fonctionnement d'un algorithme de machine learning……29

Figure 2. 2 : Apprentissage Supervisé. 30

Figure 2. 3 : Exemple de clustering et détection d'anomalies 31

Figure 2. 4 : Apprentissage Non Supervisé 32

Figure 2. 5 : Apprentissage par Renforcement 32

Figure 2. 6 : Apprentissage Semi Supervisé 33

Figure 2. 7 : Comparaison entre Machine Learning et Deep Learning. 36

Figure 2. 8 : Types de détection d'anomalies. 37

Figure 2. 9 : Principe de l'algorithme Isolation Forest. 40

Figure 2. 10 : Architecture d'un autoencodeur. 46

Figure 2. 11 : Délimitation de la région normale par One-Class SVM 50

Figure 3. 1 : Pipeline de collecte dynamique des syscalls(CICMalDroid2020)..…….....60

Figure 3. 2 : Distribution des cinq classes du dataset CICMalDroid 2020. 62

Figure 3. 3 : Heatmap de corrélation des 25 features. 64

Figure 3. 4 : t-SNE 2D structure Non Linéaire du dataset 65

Figure 3. 5 : visualisation PCA 2D. Séparabilité Benign vs Malware. 71

Figure 3. 6 : Carte de chaleur (heatmap). 79

Figure 3. 7 : Diagramme radar (radar chart). 79

Figure 3. 8 : Temps d'entraînement des six modèles en secondes……...77

Figure 3. 9 : Courbes ROC des six configurations expérimentales (IF, OC-SVM et AE) pour les approches AP1 et AP2..... 82

LISTE DES TABLEAUX

Tableau 1.1 : Evolution historique des applications mobiles et impact sur la sécurité 8

Tableau 1.2 : Comparaison des plateformes mobiles iOS et Android..... 9

Tableau 1.3 : Modèle en couches des applications Android et surfaces d'attaque associées 12

Tableau 1.4 : Architecture en couches d'iOS et mécanismes de sécurité associés..... 16

Tableau 1.5 : Classification des vulnérabilités selon OWASP mobile Top 10 et impacts associés..... 18

Tableau2.1 : hyperparamètres de l'Isolation Forest …….....42

Tableau 2.2 : hyperparamètres du One-Class SVM 51

Tableau 2.3 : comparaison des trois algorithmes de détection d'anomalies..... 53

Tableau 3.1 : Bibliothèques Python utilisées dans l'implémentation.....58

Tableau 3.2 : Distribution des classes dans CICMalDroid 2020..... 61

Tableau 3.3 : Résultats Isolation Forest (Approche 1 (Benign only))..... 73

Tableau 3.4 : Résultats One-Class SVM (Approche 1 (Benign only))..... 74

Tableau 3.5 : Résultats Autoencoder (Approche 1 (Benign only)) 75

Tableau 3. 6 : Résultats Isolation Forest (Approche 2 (SMOTETomek))..... 77

Tableau 3.7 : Résultats One-Class SVM (Approche 2 (SMOTETomek))..... 77

Tableau 3.8 : Résultats Autoencoder (Approche 2 (SMOTETomek)) 78

Tableau 3.9 : synthèse comparative des six modèles (résultats avec seuil optimal) 78

Tableau 3.10 : Comparaison avec des travaux de l'état de l'art sur CICMalDroid 2020..... 83

LISTE DES ABRÉVIATIONS

- **ACP : Analyse en Composantes Principales.**
- **AE : Autoencoder (Auto-encodeur).**
- **AOSP : Android Open-Source Project.**
- **AOT : Ahead-Of-Time.**
- **API : Application Programming Interface.**
- **APK : Android Package Kit.**
- **APT : Advanced Persistent Threat.**
- **ART : Android Runtime.**
- **ASLR : Address Space Layout Randomization.**
- **ATS : App Transport Security.**
- **AUC : Area Under the Curve (Aire sous la courbe).**
- **C&C : Command and Control.**
- **CIC : Canadian Institute for Cybersecurity.**
- **CICMalDroid : Canadian Institute for Cybersecurity Malware Android Dataset.**
- **CPU : Central Processing Unit.**
- **DBSCAN : Density-Based Spatial Clustering of Applications with Noise.**
- **DCL : Dynamic Code Loading.**
- **DEP : Data Execution Prevention.**
- **DES : Data Encryption Standard.**
- **DEX : Dalvik Executable.**
- **DGA : Domain Generation Algorithm.**
- **DL : Deep Learning.**
- **DoH : DNS over HTTPS.**
- **EDA : Exploratory Data Analysis (Analyse exploratoire des données).**
- **GMM : Gaussian Mixture Model.**
- **GPS : Global Positioning System.**
- **GPU : Graphics Processing Unit.**
- **HAL : Hardware Abstraction Layer.**
- **HTML : HyperText Markup Language.**
- **HTTPS : HyperText Transfer Protocol Secure.**
- **IA : Intelligence Artificielle.**
- **IDOR : Insecure Direct Object Reference.**

- **IF : Isolation Forest.**
- **IMEI : International Mobile Equipment Identity.**
- **IoT : Internet of Things.**
- **IPC : Inter-Process Communication.**
- **iTree : Isolation Tree.**
- **JNI : Java Native Interface.**
- **JSON : JavaScript Object Notation.**
- **K-NN : K-Nearest Neighbors.**
- **LOF : Local Outlier Factor.**
- **MASVS : Mobile Application Security Verification Standard.**
- **MD5 : Message Digest 5.**
- **ML : Machine Learning.**
- **MSE : Mean Squared Error.**
- **NDSS : Network and Distributed System Security.**
- **NFC : Near Field Communication.**
- **OC-SVM : One-Class Support Vector Machine.**
- **ONNX : Open Neural Network Exchange.**
- **OTP : One-Time Password.**
- **OWASP : Open Web Application Security Project.**
- **PR : Precision-Recall (Courbe Précision-Rappel).**
- **PWA : Progressive Web Application.**
- **RAT : Remote Access Trojan.**
- **RBF : Radial Basis Function.**
- **RC4 : Rivest Cipher 4.**
- **ReLU : Rectified Linear Unit.**
- **ROC : Receiver Operating Characteristic.**
- **SDK : Software Development Kit.**
- **SEED : Graine aléatoire.**
- **SELinux : Security-Enhanced Linux.**
- **SEP : Secure Enclave Processor.**
- **SGD : Stochastic Gradient Descent.**
- **SHAP : SHapley Additive exPlanations.**
- **SMS : Short Message Service.**
- **SMOTE : Synthetic Minority Over-sampling Technique.**

- **SQL : Structured Query Language.**
- **SVM : Support Vector Machine.**
- **t-SNE : t-distributed Stochastic Neighbor Embedding.**
- **TALN : Traitement Automatique du Langage Naturel.**
- **TLS : Transport Layer Security.**
- **UID : User Identifier.**
- **UI : User Interface.**
- **UMAP : Uniform Manifold Approximation and Projection.**
- **UX : User Experience.**
- **Wi-Fi : Wireless Fidelity.**

INTRODUCTION GÉNÉRALE :

L'évolution rapide des technologies numériques a profondément transformé notre société contemporaine, plaçant les applications mobiles au cœur des activités quotidiennes. Les smartphones, devenus des outils indispensables, hébergent aujourd'hui une multitude d'applications qui orchestrent une part considérable de nos activités : communications personnelles, transactions financières, gestion de la santé, domotique, et bien d'autres services essentiels. Cette omniprésence technologique, bien que source de progrès indéniables, engendre également de nouveaux défis en matière de sécurité numérique.

Avec plus de 3,6 milliards d'utilisateurs Android dans le monde et des millions d'applications disponibles sur les plateformes de téléchargement, l'écosystème mobile est devenu une cible privilégiée pour les cyberattaquants. Les applications malveillantes exploitent les vulnérabilités des systèmes et des logiciels afin de porter atteinte à la confidentialité, à l'intégrité et à la disponibilité des données des utilisateurs. Ces attaques prennent diverses formes : vol de données personnelles, détournement de ressources, espionnage, ransomware mobile ou encore accès non autorisé aux systèmes d'information qui menacent tant la vie privée des utilisateurs que la stabilité des infrastructures numériques.

Avec l'augmentation constante du nombre d'applications et la sophistication des techniques d'attaque, les mécanismes de sécurité traditionnels montrent progressivement leurs limites. Les approches classiques de détection, principalement basées sur les signatures ou les règles statiques, nécessitent une connaissance préalable des menaces existantes. Face à cette évolution constante des menaces, il devient impératif de développer des systèmes de détection capables d'identifier des comportements anormaux, même lorsqu'il s'agit d'attaques inconnues ou dites « zero-day ».

Dans ce contexte, l'intelligence artificielle s'impose comme une technologie prometteuse pour renforcer la sécurité des applications mobiles. Grâce aux techniques d'apprentissage automatique et d'apprentissage profond, les systèmes intelligents sont capables d'analyser de grandes quantités de données, d'identifier des modèles de comportement et de détecter automatiquement des anomalies susceptibles de révéler une activité malveillante.

Le présent travail s'inscrit dans cette problématique en proposant une approche innovante de détection d'anomalies comportementales dans les applications Android, fondée sur l'intelligence artificielle. Plus précisément, nous explorons l'utilisation de l'apprentissage non supervisé pour identifier les applications malveillantes à partir de l'analyse de leurs appels

système, sans nécessiter de connaissance préalable sur les malwares existants. Cette méthode offre l'avantage de pouvoir détecter des menaces inconnues, tout en allégeant la charge de collecte et d'étiquetage de vastes ensembles de données malveillantes.

Pour atteindre cet objectif, nous avons mis en œuvre et évalué trois algorithmes complémentaires (Isolation Forest, Autoencodeur et One-Class SVM) selon deux protocoles expérimentaux distincts : une approche one-class stricte (entraînement exclusif sur applications bénignes) et une approche semi-supervisée avec rééquilibrage SMOTETomek, sur le jeu de données de référence CICMalDroid 2020.

À travers cette étude, nous cherchons à comprendre les principaux défis liés à la cybersécurité mobile, à analyser les différentes approches de détection d'anomalies existantes, puis à proposer une solution basée sur l'intelligence artificielle adaptée aux contraintes et aux besoins des applications mobiles modernes.Cette approche vise ainsi à contribuer au développement de systèmes de sécurité intelligents, capables d'identifier automatiquement des comportements suspects et d'améliorer la détection proactive des cybermenaces dans les environnements mobiles modernes.

Structure du Mémoire

Le présent mémoire s'articule autour de trois chapitres complémentaires :

Chapitre 1 : Applications mobiles et enjeux de sécurité.

Ce chapitre pose les fondements théoriques en présentant l'écosystème des applications mobiles (définition, classification, évolution), leur architecture technique (Android et iOS), et les mécanismes de sécurité natifs. Nous y analysons les menaces et vulnérabilités selon le référentiel OWASP Mobile, définissons la notion d'anomalie comportementale, et exposons les limites des méthodes classiques de détection (signatures, règles fixes) face aux techniques d'attaque modernes (polymorphisme, métamorphisme, APT).

Chapitre 2 : Intelligence artificielle et détection d'anomalies.

Ce chapitre présente les fondements du Machine Learning et du Deep Learning appliqués à la cybersécurité mobile. Nous y présentons les types d'apprentissage automatique : supervisé, non supervisé, semi-supervisé, justifions le choix de l'apprentissage non supervisé et détaillons les trois algorithmes retenus (isolation, Autoencodeur et One-Class SVM). Pour chacun, nous

présentons les fondements théoriques, l'architecture, les hyperparamètres clés, ainsi qu'une analyse comparative de leurs avantages et limites.

Chapitre 3 : Implémentation et évaluation du système de détection d'anomalies.

Le dernier chapitre détaille la mise en œuvre expérimentale sur le jeu de données CICMalDroid 2020 (11 598 applications, 476 fonctionnalités d'appels système). Nous y présentons l'environnement de développement, l'analyse exploratoire des données ainsi les différentes étapes de prétraitement des données, notamment la normalisation, le rééquilibrage SMOTETomek et la réduction de dimensionnalité. Ce chapitre expose ensuite les protocoles expérimentaux appliqués pour les approches monoclasse et semi-supervisée. Les résultats des trois algorithmes sont analysés comparativement et confrontés à l'état de l'art. Une discussion identifie les forces de l'approche proposée, ses limites et les perspectives d'amélioration.

Une conclusion générale synthétise les contributions de ce travail et ouvre des perspectives de recherche futures.

CHAPITRE

01

APPLICATIONS MOBILES ET ENJEUX DE SECUTITE



1.1 INTRODUCTION :

Au cours des dernières années, l'utilisation des application mobiles s'impose comme un défi omniprésent incontournable, répondant à des besoins personnels ou professionnels notamment dans les domaines de l'éducation, la communication, la santé ainsi que le commerce en ligne.

Le marché mondial des applications mobiles connait une expansion sans précédent, portée par plus de 6.9 milliards d'utilisateurs de smartphones recensés à l'échelle mondiale [1]. Les plateformes mobiles représentent un écosystème d'une densité et d'une complexité sans précédent dans l'histoire de l'informatique, le Google Play Store héberge plus de 3,5 millions d'applications tandis que l'App Store d'Apple en propose plus de 1,6 million [2]. Cette dynamique illustre à la fois l'omniprésence des applications mobiles et l'enjeu économique et sécuritaire colossal qu'elles représentent.

Si ces applications offrent des possibilités extraordinaires et transforment positivement le quotidien de milliards d'individus, elles constituent également une surface d'attaque élargie et une cible privilégiée pour les cybercriminels. Chaque application installée représente un point d'entrée potentiel vers des données hautement sensibles : coordonnées bancaires, données médicales, identifiants d'authentification, localisation géographique en temps réel et contenus multimédias personnels. Cette richesse informationnelle concentrée dans des dispositifs portables, connectés en permanence et partiellement hors du contrôle des équipes de sécurité des entreprises, attire inévitablement des acteurs malveillants dont les méthodes ne cessent de se sophistiquer.

Selon le rapport Kaspersky Mobile Threat Report 2024, plus de 33,8 millions d'attaques de malwares mobiles ont été enregistrées en 2023. Une grande partie de ces attaques vise particulièrement le système Android, notamment de son caractère ouvert et de sa domination mondiale du marché. Ces menaces exploitent différentes failles, que ce soit au niveau des applications elles-mêmes, dans les mécanismes de distribution, ou dans le comportement des utilisateurs.

Dans ce contexte, la sécurité des applications mobiles s'impose comme une priorité majeure. Elle ne se limite pas uniquement à la protection de la vie privée des utilisateurs, mais concerne également la fiabilité, la crédibilité et la durabilité des entreprises. Il devient donc essentiel de mettre en œuvre des mécanismes de sécurité solides afin de garantir la confidentialité, l'intégrité et la disponibilité des données.

Cependant, face à l'évolution constante des menaces, les approches traditionnelles de détection, fondées sur les bases de signatures virales et les règles comportementales statiques montrent des limites structurelles face aux malwares polymorphes, aux attaques zero-day et aux techniques d'évasion avancées. L'intelligence artificielle, et plus particulièrement les méthodes d'apprentissage automatique non supervisé, offrent une alternative prometteuse : celle de la détection d'anomalies comportementales, capable d'identifier des menaces inconnues sans connaissance préalable de leurs signatures.

Ce chapitre pose les fondations conceptuelles et techniques de notre travail. Nous définissons et classifions les applications mobiles, analysons leur architecture interne pour identifier les surfaces d'attaque, dressons un panorama des menaces et des taxonomies de malwares en particulier celles du jeu de données CICMalDroid 2020 qui constitue le support expérimental de ce mémoire, formalisons la notion d'anomalie comportementale, et démontrons enfin les limites des approches classiques qui motivent notre choix méthodologique.

1.2 FONDEMENTS DES APPLICATIONS MOBILES :

1.2.1 Définition et Évolution Historique :

Dans le domaine informatique, une application mobile est un logiciel développé et conçu pour s'exécuter nativement sur un appareil informatique portable (smartphone, tablette, montre connectée) équipé d'un système d'exploitation mobile dédié. Elle tire parti des ressources matérielles spécifiques à ces appareils capteurs inertiels (accéléromètre, gyroscope), système de positionnement (GPS), interfaces de communication sans fil (Wi-Fi, Bluetooth, NFC), caméra, microphone, généralement distribuée via une plateforme de téléchargement officielle soumise à des processus de validation et de signature cryptographique [3]

Les applications mobiles se distinguent des logiciels de bureau par quatre propriétés fondamentales qui déterminent à la fois leur valeur d'usage et leur surface d'exposition aux menaces : leur portabilité (disponibilité en tout lieu), leur connectivité permanente (accès continu aux réseaux), leur accès aux capteurs physiques (données biométriques, localisation, environnement sonore et visuel), et leur intégration dans des écosystèmes commerciaux fortement régulés (stores, abonnements, achats in-app). Ces quatre propriétés en font à la fois des outils d'une valeur pratique considérable et des cibles privilégiées pour les attaquants cherchant à collecter des données personnelles ou à compromettre des systèmes d’information [4].

La figure 1.1 illustre les trois principales plateformes de distribution des applications mobiles.

- L'App Store (plateforme d'Apple)
- Le Google Play (plateforme de google)
- Le Windows Phone Store (plateforme de Microsoft)

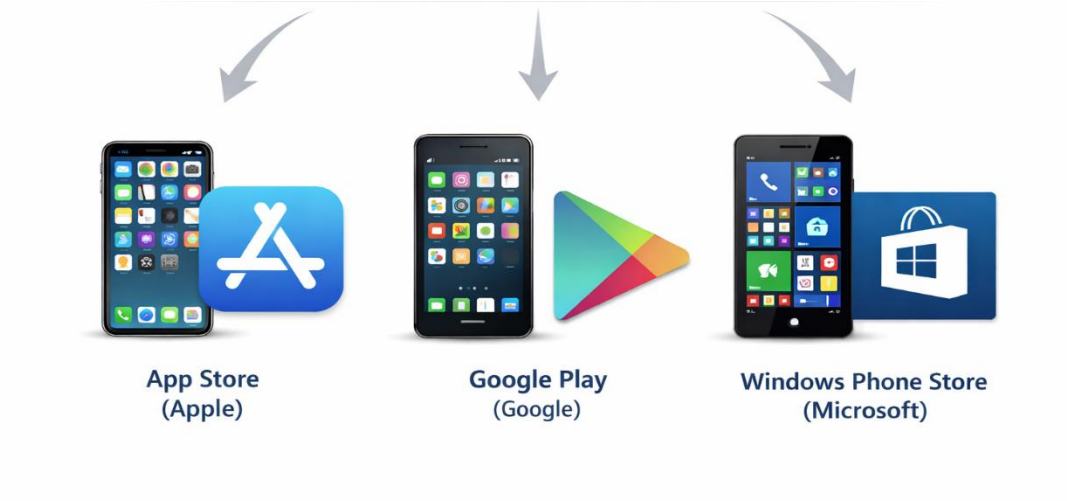


Figure 1. 1: Plateformes de distribution des applications mobiles (Généré par intelligence artificielles ,2026)

1.2.1.1 Évolution historique des applications mobiles :

Le tableau 1.1 représente l'histoire des applications mobiles s'articule en cinq phases distinctes, chacune caractérisée par des ruptures technologiques majeures ayant des implications directes sur le paysage de la sécurité mobile :

Période	Événement technologique majeur	Langage/ Technologie	Impact sur la sécurité
2007	Lancement de l'iPhone (Apple) début de l'ère smartphone moderne	Objective-C	Modèle d'application fermé, contrôle éditeur fort
2008	Création App Store (Apple) et Android Market (Google)	Java (Android), Obj-C (iOS)	Premier écosystème de distribution contrôlé et signé

2010–2015	Explosion applicative (Instagram, WhatsApp) et l'amélioration du multiplateforme (design, sécurité)	HTML5, frameworks cross-platform	Prolifération des vecteurs d'attaque, premier adwares mobiles
2015-2020	Spécialisation sectorielle (santé finance), IA embarquée	Kotlin, Swift, React Native, Flutter	Malwares ciblés (banking tojans), APT mobiles
2020-2026	Super-app, IoT mobile, IA générative, edge computing	Rust, Kotlin Multiplatform, ONNX	Zero-day exploits, attaques sur modèles IA embarqués

Tableau 1.1: Evolution historique des applications mobiles et impact sur la sécurité

1.3 CLASSIFICATION DES APPLICATIONS MOBILES :

Les applications mobiles peuvent être classées selon trois paradigmes architecturaux de développement, chacun présentant des compromis distincts en matière de performance, de maintenabilité et de sécurité. La compréhension de ces différences est essentielle pour analyser les surfaces d'attaque spécifiques à chaque catégorie.

1.3.1 Applications natives :

Les applications natives sont développées spécifiquement pour une plateforme cible unique (Android ou iOS). Elles utilisent le kit de développement officiel SDK (Software Development Kit) et des langages de programmation natifs associés (Java, Kotlin ou le langage objective-C) fournis par les éditeurs de plateformes.[5]. Elles bénéficient d'un accès complet et optimisé à l'ensemble des APIs système et ressources matérielles. Le tableau 1.2 présente une comparaison des deux principales plateformes mobiles iOS et Android selon des critères techniques, sécuritaires et commerciaux.

Caractéristique	iOS (Apple)	Android (Google)
Langages principaux	Swift, Objective-C	Kotlin, Java

Environnement de développement	Xcode	Android studio
Interface utilisateur	Simple, fluide et uniforme	Flexible, varie selon le fabricant
Modèle de sécurité	Très élevé (système fermé, sandbox strict, signature obligatoire)	Élevé, mais plus exposé (open-source, sideloading possible)
Part de marché mondial (2024)	~28% (segment premium)	~72% (adoption mondiale)

Tableau 1.2 : Comparaison des plateformes mobiles iOS et Android

- **Les Avantages :**

Les applications natives sont spécifiquement optimisées pour chaque système d'exploitation mobile (iOS, Android, Windows Phone, BlackBerry)**,** ce qui leur permet d'exploiter pleinement les capacités de l'appareil. Cette optimisation se traduit par une interface utilisateur fluide, rapide et réactive, améliorant ainsi l'expérience et l'efficacité d'utilisation. Par ailleurs, elles offrent un accès complet aux fonctionnalités natives du dispositif (GPS, contacts, agenda, etc.) et possibilité l'utilisation hors connexion (la continuité de service).

Sur le plan sécuritaire les applications natives bénéficient d'un accès complet aux APIs de sécurité de la plateforme. Elles peuvent utiliser des solutions de stockage sécurisé comme le Trousseau sur iOS ou le Keystore sur Android, Secure Enclave pour les opérations cryptographiques isolées, et APIs biométriques natives (Touch ID, Face ID, Android Biometric). La signature de code obligatoire assure l'intégrité de l'application distribuée et permet la traçabilité du développeur.

- **Les inconvénients :**

Malgré leurs nombreux avantages, les applications natives présentent également certaines limites. Leur développement implique un coût relativement élevé, puisqu'il est nécessaire de créer une version distincte pour chaque système d'exploitation. De plus, les processus d'analyse et de maintenance peuvent s'avérer complexes, notamment en raison de la diversité des

plateformes et des mises à jour fréquentes. Enfin, le temps de développement est généralement plus long, ce qui peut ralentir la mise sur le marché de l'application.

Le maintien de deux bases de code distinctes (iOS et Android) augmente la surface de risque globale et accroît le délai de correction des vulnérabilités découvertes (patch latency). La dépendance aux APIs propriétaires peut par ailleurs introduire des vulnérabilités spécifiques à chaque version de l'OS.

1.3.2 Applications Web Progressives (PWA) :

Une application web progressive (PWA) représente une amélioration d'une application web traditionnelle. Elle combine les avantages des technologies web et des applications natives. Les PWA peuvent fonctionner via un navigateur web et même hors connexion grâce aux Service Workers. Elles reposent sur trois piliers techniques [6] : les Service Workers (scripts s'exécutant en arrière-plan, indépendamment de la page web, permettant le mode hors-ligne et les notifications push), le Web App Manifest (fichier JSON déclarant les métadonnées de l'application pour l'installation sur l'écran d'accueil) et l'API Cache (gestion locale des ressources réseau).

- **Les avantages :**

Les applications web progressives (PWA) présentent plusieurs avantages notables, notamment leur capacité à fonctionner sur différentes plateformes à partir d'un code unique, qu'il s'agisse d'Android, d'iOS ou même d'un ordinateur. Cette approche permet de réduire considérablement les coûts de développement par rapport aux applications natives. De plus, les mises à jour peuvent être déployées instantanément, sans passer par les plateformes de téléchargement, ce qui simplifie leur gestion. L'installation ainsi que la maintenance de ces applications sont également plus simples, offrant une grande flexibilité

- **Les inconvénients :**

Cependant, ces applications présentent aussi certaines limites. L'accès aux fonctionnalités matérielles reste restreint et dépend fortement du système d'exploitation utilisé. Par ailleurs, elles offrent généralement une expérience utilisateur moins fluide et une intégration plus limitée avec les composants natifs du système contrairement aux applications natives. Enfin, leurs performances sont souvent inférieures, ce qui les rend moins adaptées aux applications nécessaires

1.3.3 Applications Hybrides :

Les applications hybrides combinent une approche intermédiaire entre les applications web et les applications natives afin de permettre un développement multiplateforme à partir d'une base de code unique en technologies web (HTML5, CSS3, JavaScript/TypeScript), elles sont encapsulées dans un conteneur natif permettant leur distribution via les stores officiels et leur accès partiel aux ressources système via des plugins. Les frameworks dominants sont React Native (Meta), Flutter (Google) et Ionic / Capacitor [7], qui facilitent la création d'une application unique compatible et bon équilibrer entre performance et flexibilité.

- **Les avantages :**

Les applications hybrides permettent de développer une seule base de code compatible avec plusieurs plateformes, ce qui réduit considérablement les coûts et le temps de développement. Elles facilitent également la maintenance grâce à une gestion centralisée, tout en offrant une solution pratique pour déployer rapidement une application sur différents systèmes.

- **Les inconvénients :**

Cependant, elles présentent certaines limites, notamment en termes de performances et d'optimisation, car elles sont moins adaptées aux spécificités de chaque plateforme. Leur dépendance aux frameworks et aux plugins tiers peut également introduire des vulnérabilités et augmenter la surface d'attaque. De plus, leur intégration avec les fonctionnalités natives reste limitée, ce qui peut affecter l'expérience utilisateur.

1.4 ARCHITECTURE DE l'APPLICATION MOBILE :

L'architecture interne d'une application mobile détermine fondamentalement sa posture de sécurité. Comprendre cette architecture en détail est indispensable pour identifier les vecteurs d'attaque exploitables, comprendre les mécanismes de protection en place, et concevoir des systèmes de détection d'anomalies comportementales pertinents. Nous présentons ici les architectures des deux plateformes dominantes Android et iOS avec un accent particulier sur Android, qui constitue la base du jeu de données CICMalDroid 2020 utilisé dans ce mémoire.

1.4.1 Architecture Logicielle en Couches :

Indépendamment de la plateforme cible, les applications mobiles s'organisent selon un modèle architectural en couches (*layered architecture*), où chaque couche possède une responsabilité fonctionnelle clairement définie et interagit avec les couches adjacentes selon

des interfaces strictement définies. Cette architecture est essentielle pour comprendre le fonctionnement des applications et identifier les vecteurs d'attaque potentiels qui mettre en place des mécanismes de détection d'anomalies efficaces. Le tableau 1.3 présente le modèle en couches des application Android en associant à chaque couche ses responsabilités, ses composants clés et les principaux vecteurs d'attaque qui lui sont associés.

Couche	Responsabilité	Composants clés (Android)	Vecteurs d'attaque principaux
Présentation (UI/UX)	Interface utilisateur, gestion des interactions tactiles	Activities, Fragments, Jetpack Compose, UIViews	Injection d'interface, tapjacking, overlay attacks
Logique métier	Traitement des données, règles fonctionnelles	ViewModels, Use Cases, Services Android	Manipulation de logique, contournement contrôles accès
Accès aux données	Persistance locale, synchronisation distante	Room/SQLite, Retrofit, OkHttp, DataStore	Injection SQL, interception réseau, vol de tokens
Système / Noyau	Ressources matérielles, OS, sécurité bas niveau	Linux Kernel, HAL, Binder IPC	Escalade de privilèges, exploits kernel, rootkits

Tableau 1. 3: Modèle en couches des applications Android et surfaces d'attaque associées

1.4.2 Architecture de la Plateforme Android :

Android constitue la plateforme mobile dominante avec environ 72 % des parts de marché mondial en 2024. Son architecture en cinq couches, documentée officiellement par Google [8], détermine l'ensemble des mécanismes de sécurité et des surfaces d'attaque qui nous intéressent

dans le contexte de la détection de malwares. Nous détaillons chaque couche en soulignant ses implications sécuritaires.

1.4.2.1 Noyau Linux (Linux Kernel) :

La couche la plus basse de l'architecture Android est un noyau Linux modifié et optimisé pour les contraintes des appareils mobiles (gestion de l'énergie, pilotes matériels spécialisés, Binder IPC). Il assure la gestion des ressources matérielles (mémoire, processeur, E/S) et constitue la fondation du modèle de sécurité Android. Chaque application s'exécute sous un identifiant utilisateur Linux unique (UID), ce qui crée une isolation au niveau du système de fichiers mécanisme dit de sandboxing applicatif. Cette isolation empêche structurellement une application d'accéder directement aux fichiers ou aux ressources mémoire d'une autre application sans mécanisme explicite de partage.

Du point de vue sécuritaire, des vulnérabilités dans les pilotes du noyau (pilotes GPU, audio, caméra) constituent des vecteurs d'exploitation particulièrement critiques car ils s'exécutent avec des privilèges élevés. Les exploits de type privilege escalation ciblant le noyau Linux permettent à un attaquant de sortir du sandbox applicatif et d'obtenir un accès système complet (root).

1.4.2.2 Couche d'abstraction matérielle (HAL « Hardware Abstraction Layer ») :

La HAL fournit des interfaces standardisées entre les pilotes matériels propriétaires des fabricants et le framework Android supérieur. Elle permet à différents fabricants (Samsung, Qualcomm, MediaTek) d'implémenter leurs propres pilotes sans modifier le code AOSP (Android Open Source Project). Sécuritairement, les implémentations HAL défaillantes représentent une surface d'attaque critique, car elles s'exécutent avec des privilèges élevés et traitent directement les données des capteurs matériels sensibles (caméra, microphone, capteurs biométriques).

1.4.2.3 Android Runtime (ART) et format DEX :

L'Android Runtime (ART), introduit avec Android 5.0 Lollipop, remplace l'ancienne machine virtuelle Dalvik et adopte une stratégie de compilation mixte AOT (Ahead-Of-Time) et JIT (Just-In-Time), améliorant significativement les performances d'exécution et réduisant la consommation énergétique. Les applications Android sont compilées au format DEX (Dalvik EXecutable), un bytecode optimisé pour les contraintes mémoire des appareils mobiles [4].

Ce format DEX est central dans l'analyse de sécurité des applications : des outils comme Jadx ou dex2jar permettent de décompiler les fichiers DEX en code Java lisible, facilitant le reverse engineering — aussi bien pour les chercheurs en sécurité que pour les attaquants cherchant à analyser des applications concurrentes, à extraire des clés cryptographiques codées en dur, ou à réemballer des applications légitimes avec du code malveillant (repackaging attack).

1.4.2.4 Application Framework et composants Android :

L'Application Framework fournit l'ensemble des APIs Java utilisées par les développeurs. Du point de vue sécuritaire, les quatre composants fondamentaux d'une application Android (Activity, Service, BroadcastReceiver, ContentProvider) représentent des surfaces d'exposition distinctes :

- **Activity** : gère le cycle de vie des écrans. Des vulnérabilités dans la gestion des Intents (notamment les Intents implicites) peuvent permettre des attaques par Intent Hijacking, où une application malveillante intercepte des Intents destinés à une application légitime.
- **Service** : composant s'exécutant en arrière-plan, sans interface utilisateur. Les malwares exploitent abondamment les Services pour maintenir leurs activités malveillantes (exfiltration de données, cryptominage, surveillance) de manière persistante, même lorsque l'application principale n'est pas visible.
- **BroadcastReceiver** : composant recevant des messages diffusés par le système ou d'autres applications. Des Receivers non protégés par des permissions adéquates peuvent être déclenchés par des applications malveillantes, ou intercepter des Intents broadcast sensibles (comme SMS_RECEIVED pour les banking trojans).
- **ContentProvider** : mécanisme de partage de données structurées entre applications. Une exposition involontaire (exported=true sans permission appropriée) peut conduire à des fuites massives de données utilisateurs accessibles à toute application installée sur l'appareil.

1.4.2.5 Modèle de sécurité Android : sandbox, permissions et SELinux :

Le système Android intègre plusieurs mécanismes de sécurité pour protéger les applications et les données. Chaque application s'exécute dans un environnement isolé (sandbox) avec un identifiant unique [4], distinctif de l'accès direct aux données des autres applications. L'accès aux ressources sensibles est contrôlé par un système de permissions nécessitant l'accord de l'utilisateur. De plus, les applications doivent être signées numériquement afin de garantir leur intégrité. Android renforce également sa sécurité grâce à

des mécanismes avancés comme SELinux, qui limitent les actions des processus, et Verified Boot, qui assurent l'intégrité du système au démarrage. Enfin, le service Google Play Protect permet de détecter en continu les applications malveillantes à l'aide d'analyses comportementales. Ce modèle de sécurité, bien que robuste, peut être contourné par des applications malveillantes qui exploitent des vulnérabilités dans le système ou abusent de permissions légitimes accordées par l'utilisateur.

1.4.3 Architecture de la Plateforme iOS :

iOS adopte une philosophie architecturale fondamentalement différente d'Android, caractérisée par un niveau de fermeture et de contrôle éditorial nettement supérieur. Cette philosophie de walled garden se traduit par un modèle de sécurité par défaut plus strict, au prix d'une flexibilité réduite pour les développeurs et les utilisateurs [9]. Le tableau 1.4 représente l'architecture et le mécanisme de sécurité d'iOS.

Couche Ios	Rôle principal	Frameworks/ Technologies	Mécanismes de sécurité spécifiques
Cocoa Touch	Interface utilisateur, gestures, cycle de vie des apps	UIKit, SwiftUI, Foundation, EventKit	App Transport Security (ATS), entitlements système
Media	Traitement audio, vidéo, graphisme 2D/3D	AVFoundation, Core Graphics, Metal, SceneKit	Accès conditionnel via système de permissions iOS
Core Services	Services fondamentaux (données, réseau, cloud)	Core Data, CloudKit, StoreKit, Core Location	Keychain Services, Data Protection API (chiffrement fichiers)

Core OS	Noyau XNU, sécurité système, cryptographie bas-niveau	XNU Kernel, libSystem, Security Framework	ASLR, DEP/XD, Secure Enclave Processor (SEP), codesign

Tableau 1.4: Architecture en couches d'iOS et mécanismes de sécurité associés

1.4.3.1 Mécanismes de sécurité iOS avancés :

iOS intègre plusieurs technologies de protection absentes ou optionnelles sur Android :

- **ASLR (Address Space Layout Randomization) :** randomise les adresses mémoire des processus à chaque exécution, rendant l'exploitation de vulnérabilités de type buffer overflow significativement plus difficile.
- **DEP / XD (Data Execution Prevention / Execute Disable) :** marque les pages mémoire de données comme non-exécutables, prévenant l'injection de shellcode en mémoire.
- **Secure Enclave Processor (SEP) :** coprocesseur cryptographique physiquement isolé du processeur principal (Application Processor). Il gère les clés biométriques (Touch ID / Face ID) et les opérations de chiffrement critiques sans que ces données n'exposent jamais le domaine principal.
- **Code Signing obligatoire :** toutes les applications doivent être signées avec un certificat Apple valide issu du programme de développement officiel, éliminant pratiquement la distribution d'applications non vérifiées en dehors des contextes de jailbreak.
- **App Sandbox + Entitlements :** chaque application est confinée dans un bac à sable strict avec un répertoire de données privé. L'accès à des fonctionnalités système étendues (notifications push, accès réseau étendu, accès aux données de santé) nécessite l'octroi d'entitlements explicites par Apple.

1.5 SECURITE DES APPLICATIONS MOBILES :

1.5.1 Menaces et Vulnérabilités (Référentiel OWASP Mobile) :

La sécurité des applications mobiles est mise à l'épreuve par un ensemble varié de menaces, dont la classification systématique est indispensable pour organiser une réponse défensive efficace. **L'OWASP** (Open Web Application Security Project) maintient le référentiel OWASP Mobile Application Security Verification Standard **(MASVS)** [10], qui constitue la référence académique et industrielle pour l'évaluation de la sécurité des applications mobiles. Le tableau

1.5 représente le classement OWASP Mobile Top 10 recense les catégories de vulnérabilités les plus critiques.

Menaces	Vulnérabilité	Description technique	Impact potentiel
M1	Utilisation incorrecte de la Plateforme	Mauvaise utilisation des APIs Android/iOS : permissions excessives, Intents implicites mal protégés, exposition de Content Providers	Fuite de données, escalade de privilèges
M2	Stockage non sécurisé des données	Conservation en clair de données sensibles (identifiants, tokens) dans SharedPreferences, SQLite, logs système ou fichiers externes	Vol d'identifiants, tokens d'authentification
M3	Communication réseau non sécurisée	Absence ou mauvaise implémentation TLS/SSL, validation de certificats incorrecte, cleartext traffic	Attaques Man-in-the-Middle (MITM), interception
M4	Authentification et autorisation insuffisantes	Absence de vérification côté serveur, tokens persistants non invalidés, authentification côté client uniquement	Usurpation d'identité, accès non autorisé
M5	Cryptographie insuffisante	Algorithmes obsolètes (MD5, DES, RC4), clés hardcodées dans le code source, vecteurs d'initialisation réutilisés	Déchiffrement des données protégées

M6	Autorisation insuffisante (IDOR)	Contrôles d'accès absents ou contournables côté serveur, Insecure Direct Object Reference	Accès à des ressources et données non autorisées
M7	Qualité du code client	Buffer overflows, injection SQL locale, vulnérabilités de format string dans le code natif (JNI)	Exécution de code arbitraire
M8	Falsification du code	Apps repackagées avec code malveillant, contournement du certificate pinning, jailbreak/root detection bypass	Distribution de malwares signés, contournement protections
M9	Reverse Engineering	Décompilation via Jadx/APKTool, extraction de clés et algorithmes propriétaires, analyse des protocoles	Vol de propriété intellectuelle, clés cryptographiques
M10	Fonctionnalités superflues	Backdoors de débogage non désactivé en production, endpoints de test exposés, logs verbeux	Accès non intentionnel aux systèmes et données

Tableau 1.5: Classification des vulnérabilités selon OWASP mobile Top 10 et impacts associés

1.5.1.1 Vecteurs d'attaque spécifiques à Android :

Au-delà des vulnérabilités génériques du OWASP Top 10, l'écosystème Android présente des vecteurs d'attaque spécifiques liés à son architecture ouverte et à la diversité de ses implémentations matérielles :

- **Intent Hijacking :** interception d'Intents implicites par des applications malveillantes déclarant des filtres d'Intent similaires. Un attaquant peut ainsi intercepter des échanges de données entre applications légitimes (partage de fichiers, authentification OAuth).
- **Tapjacking / Overlay Attacks :** affichage d'une couche transparente par-dessus une application légitime pour capturer les clics de l'utilisateur. Utilisé notamment par les banking trojans pour superposer de fausses interfaces de saisie de credentials bancaires.
- **Abus des Accessibility Services :** détournement des APIs d'accessibilité Android (initialement conçues pour les utilisateurs handicapés) pour simuler des interactions utilisateur, effectuer des captures d'écran, lire le contenu de l'écran et extraire des données sensibles affichées par d'autres applications.
- **Dynamic Code Loading (DCL) :** téléchargement et exécution de code compilé au moment de l'exécution. Cette technique permet de contourner l'analyse statique lors de la soumission au store, en dissimulant le payload malveillant dans du code chargé dynamiquement après installation.
- **Sideloading malveillant :** installation d'APK provenant de sources non officielles (sites tiers, partage peer-to-peer), contournant les mécanismes de vérification du Google Play Store. Cette pratique représente le principal vecteur de distribution des banking trojans dans les marchés émergents.

1.5.2 Taxonomie des Malwares Mobiles :

Les malwares mobiles sont définis comme tout programme ou logiciel informatique conçu délibérément pour compromettre l'intégrité, la confidentialité ou la disponibilité d'un appareil mobile et de ses données [11]. Dans l'écosystème Android, ces menaces ont évolué rapidement, passant de nuisances relativement mineures (adwares simples) à des outils d'espionnage sophistiqués utilisés dans des opérations de cyberguerre et d'espionnage industriel.

1.5.3 Attaques courantes (exfiltration de données, comportements anormaux.) :

L'analyse des applications mobiles malveillantes ne se limite pas à leur classification, mais repose également sur l'identification des primitives d'attaque qu'elles implémentent. Au-delà de la catégorisation des malwares, il convient d'identifier les primitives d'attaque les plus fréquemment observées lors d'analyses comportementales [13,14] :

- **L'exfiltration de données** : il s'agit du transfert non autorisé d'information ou de données sensibles (données bancaires, contacts, identifiants, etc.) vers un serveur de commande et de contrôle souvent chiffrée via HTTPS.
- **L'escalade de privilèges** : consiste à obtenir des droits d'accès supérieurs à ceux initialement accordés en exploitant des vulnérabilités du système.
- **Le téléchargement dynamique de code** : exécution de code supplémentaire au moment de l'exécution pour éviter les analyses statiques.
- **La persistance** : mécanismes assurant la survie du malware sur le dispositif après redémarrage ou tentatives de désinstallation notamment via l'enregistrement de services ou la modification des paramètres système.
- **La cryptominage abusive (Cryptojacking):** utilisation des ressources matérielles (CPU, GPU, batterie) de l'appareil pour le minage de crypto-monnaies sans consentement, entraînant une dégradation des performances et une surchauffe du dispositif.

1.6 NOTION D'ANOMALIE DANS LES APPLICATIONS MOBILES :

1.6.1 Définition d'une anomalie :

Dans le domaine de la détection des intrusions, une anomalie est définie comme tout écart significatif et inhabituel par rapport à un comportement de référence établi comme normal [15].

Dans le contexte spécifique des applications mobiles, le comportement « normal » d'une application correspond à l'ensemble des actions qu'elle est censée exécuter conformément à sa fonction déclarée : accès aux ressources nécessaires à son fonctionnement, communications réseau conformes à son usage, consommation des ressources proportionnelle à ses opérations. Une anomalie surgit lorsqu'une application s'écarte de ce comportement attendu, notamment en accédant à des ressources non justifiées par son usage ou en générant des événements système inhabituels [16].

1.6.2 Exemples Concrets d'Anomalies Comportementales :

Pour ancrer la notion d'anomalie dans la réalité du jeu de données CICMalDroid 2020, nous présentons des scénarios représentatifs directement liés aux cinq classes de l'ensemble de données [17,18] :

- Une application de calculatrice accédant répétitivement aux contacts et établissant des connexions vers des serveurs distants, constituant un comportement anormal par rapport à sa fonction déclarée.

- Un jeu mobile générant une activité réseau intensive durant la période nocturne, en mode veille, pouvant indiquer la présence d'un comportement de type botnet.
- Une application de lampe de poche sollicitant l'activation du microphone, ce qui représente une demande de permission disproportionnée par rapport à ses fonctionnalités légitimes.
- Une augmentation soudaine de la consommation CPU atteignant 90–99 % alors que l'application est inactive, caractéristique typique d'un cryptomineur mobile.
- L'envoi automatisé de SMS à des numéros surtaxés sans interaction utilisateur, comportement typique des SMS Trojans du jeu de données CICMalDroid 2020 [12].

1.7 LIMITES DES METHODES CLASSIQUES DE DETECTION :

Une compréhension approfondie des limites des approches traditionnelles de détection de malwares est essentielle pour justifier, de manière objective, l'adoption d'une approche basée sur l'apprentissage non supervisé. Cette section examine de manière rigoureuse les insuffisances des méthodes classiques face à l'évolution du paysage actuel des menaces mobiles.

1.7.1 Détection par Signatures (Antivirus Traditionnels) :

Les antivirus mobiles traditionnels sont des logiciels qui détectent les logiciels malveillants, reposent sur la comparaison d'une empreinte numérique (hash) avec une base de signatures connues [19] : si l'empreinte d'un fichier examiné correspond à une signature répertoriée, le fichier est classifié comme malveillant et l'action de remédiation configurée est déclenchée. Si cette approche reste efficace pour détecter les malwares connus et répertoriés, elle présente des limitations fondamentales dans un contexte où les malwares évoluent en permanence :

- Incapacité à détecter les nouvelles variantes (malwares zero-day) ne figurant pas encore dans les bases de signatures.
- Échec face aux techniques d'obfuscation de code qui modifient l'empreinte du malware sans altérer sa fonctionnalité.
- Dépendance à des mises à jour fréquentes, créant une fenêtre de vulnérabilité entre la découverte d'un nouveau malware et sa prise en compte dans la base de signatures.[10]

1.7.2 Systèmes de Détection à Règles Fixes :

Les systèmes de détection basés sur des règles fixes (rule-based systems) tentent de capturer le comportement malveillant à travers des conditions définies par des experts en sécurité. Ces règles peuvent prendre la forme de politiques de permission, de patterns d'appels système, ou

de conditions sur le trafic réseau [20] (ex : si un programme modifie le registre + envoie des données → alerte).

Bien que plus flexibles que les antivirus à signatures, ces systèmes souffrent eux aussi de limitations importantes :

- **Rigidité conceptuelle et obsolescence rapide :** les règles sont définies par des experts humains sur la base des menaces connues au moment de leur rédaction. Dès qu'une nouvelle technique d'attaque émerge, les règles existantes deviennent partiellement ou totalement inefficaces. La cadence d'évolution des malwares mobiles dépasse largement la capacité humaine de mise à jour des règles.
- **Taux de faux positifs structurellement élevés :** une règle trop générale risque de classifier des applications légitimes comme malveillantes. Par exemple, une application de monitoring système utilisée par les administrateurs réseau peut déclencher des règles conçues pour détecter les spywares en raison de ses accès légitimes aux ressources système — générant une fatigue d'alerte (*alert fatigue*) qui conduit les opérateurs à ignorer les alertes, y compris les vraies.
- **Contournement trivial par les attaquants :** des règles trop spécifiques peuvent être facilement contournées par des attaquants qui connaissent ou devinent les critères de détection. Il suffit de modifier légèrement le comportement du malware (ajout d'un délai, modification du seuil de volume réseau, fragmentation des communications) pour échapper à la règle.
- **Incapacité à gérer la complexité multidimensionnelle :** les règles manuelles ne peuvent pas capturer efficacement des patterns comportementaux complexes impliquant des dizaines ou centaines de dimensions corrélées simultanément. Les algorithmes d'apprentissage automatique, notamment les méthodes basées sur la détection d'anomalies dans des espaces de haute dimension, excellent précisément là où les règles échouent.
- **Coût de maintenance prohibitif :** l'actualisation continue d'un corpus de règles de qualité nécessite une veille permanente sur les nouvelles menaces, une expertise humaine spécialisée rares et coûteuse, ainsi que des cycles de test et de validation importants pour éviter les régressions (nouvelles règles introduisant de nouveaux faux positifs ou cassant des règles existantes).

1.7.3 Limites face aux Technique d'Attaques Modernes :

Au-delà des limitations intrinsèques de chaque méthode classique, l'émergence de techniques d'attaque avancées rend ces approches structurellement insuffisantes face au paysage des menaces mobiles contemporaines [21][22]

1.7.3.1 Polymorphisme et métamorphisme :

Les malwares polymorphes modifient dynamiquement leur représentation binaire (en chiffrant leur payload avec une clé différente à chaque instance, générée localement sur l'appareil cible) tout en préservant leur fonctionnalité malveillante. Les malwares métamorphiques vont plus loin en restructurant intégralement leur code substitution d'instructions équivalentes, réécriture des blocs de contrôle de flux, génération de code mort ne partageant aucune séquence de bytes caractéristique entre deux instances distinctes. Ces deux techniques rendent structurellement inutile toute approche fondée sur la comparaison de signatures fixes.

1.7.3.2 Évasion par détection de l'environnement d'analyse :

Les malwares sophistiqués intègrent des mécanismes de détection des environnements d'analyse (émulateurs, sandboxes d'analyse automatisée, machines virtuelles). En identifiant des caractéristiques propres à ces environnements contrôlés telles que l'absence d'accéléromètre physique, des performances CPU inhabituelles, l'absence d'historique d'appels ou de SMS, ou encore un IMEI générique, ils adoptent un comportement bienveillant pendant la phase d'analyse, puis révèlent leur nature malveillante uniquement sur les appareils réels des victimes. Cette technique, dénommée sandbox evasion, permet d'éviter à la fois les analyses statiques et dynamiques menées dans des environnements de test standard.

1.7.3.3 Menaces Persistantes Avancées mobiles (APT) :

Les Advanced Persistent Threats (APT) ciblant les plateformes mobiles comme les malwares Pegasus, Chrysaor ou Hermit développés par des acteurs étatiques, présentent des comportements d'exécution délibérément lents et furtifs. En maintenant un niveau d'activité minimal et en s'activant uniquement lors de fenêtres temporelles spécifiques ou en réponse à des commandes distantes chiffrées, ces malwares restent systématiquement sous les seuils de détection des règles comportementales fixes et des systèmes d'analyse heuristique classiques.

1.7.3.4 Chiffrement intégral du trafic malveillant :

L'adoption généralisée du protocole TLS 1.3 par les malwares mobiles modernes rend l'inspection du contenu des communications réseau largement inopérante sans déchiffrement actif (HTTPS inspection). Une telle démarche s'avère techniquement difficile et pose également des enjeux légaux lorsqu'elle concerne des appareils mobiles personnels. Par ailleurs, l'utilisation de techniques telles que DNS over HTTPS (DoH) et les algorithmes de génération de domaines (DGA) permet aux malwares de dissimuler leurs communications avec leurs serveurs de commandement et de contrôle (C&C) dans le cadre d'un trafic HTTPS apparemment légitime, contournant ainsi efficacement les systèmes de détection basés sur des listes noires d'URLs ou d'adresses IP.

Ces constats justifient l'exploration d'approches alternatives, fondées sur l'apprentissage automatique non supervisé. Contrairement aux méthodes précédemment évoquées, ces approches ne nécessitent pas de connaissance préalable des menaces : elles identifient les anomalies comportementales par écart au modèle appris, ce qui leur confère une capacité potentielle de détection générique, y compris pour des malwares inconnus.

1.8 CONCLUSION

Ce premier chapitre a établi les fondations conceptuelles, architecturales et sécuritaires nécessaires à la compréhension et à la justification de notre approche de détection d'anomalies dans les applications mobiles par l'intelligence artificielle.

Nous avons d'abord posé une définition rigoureuse des applications mobiles et retracé leur évolution historique, soulignant comment chaque rupture technologique depuis l'iPhone de 2007 jusqu'aux super-applications intégrant de l'IA générative, a simultanément élargi les possibilités d'usage et amplifié la surface d'exposition aux menaces. La classification en trois paradigmes architecturaux : natif, web progressif et hybride a révélé que les compromis entre performance et flexibilité s'accompagnent toujours de compromis sécuritaires qu'il convient d'identifier explicitement.

L'analyse approfondie des architectures Android et iOS a permis de cartographier précisément les mécanismes de protection (sandboxing, système de permissions, SELinux, ART/DEX, Secure Enclave, ASLR, DEP) et les surfaces d'attaque exploitables. Cette connaissance architecturale conditionne directement la pertinence du choix des appels système comme feature representation : seule une compréhension fine de la couche HAL, de l'ART et

du framework applicatif permet d'apprécier pourquoi les vecteurs de fréquences de syscalls constituent une représentation comportementale à la fois exhaustive et résistante à l'obfuscation.

Le panorama des menaces mobiles, structuré selon le référentiel OWASP Mobile Top 10 et de la taxonomie en cinq classes du jeu de données CICMalDroid 2020 (Adware, Banking, SMS, Riskware, Bénigne), met en évidence la diversité et la sophistication croissante des vecteurs d'attaque Android. L'identification des primitives d'attaque observables (exfiltration, escalade de privilèges, chargement dynamique de code, overlay attacks, interception SMS) constitue le pont conceptuel entre la taxonomie des malwares et la représentation comportementale par appels système exploitée dans notre approche.

La formalisation de la notion d'anomalie comportementale, selon la taxonomie proposée par Chandola, Banerjee et Kumar (2009), qui distingue les anomalies ponctuelles, contextuelles et collectives, a fourni un cadre théorique précis sur lequel repose notre approche expérimentale. La justification du choix des vecteurs d'appels système comme espace de représentation a démontré que cette modélisation capture le comportement effectif à l'exécution, indépendamment des techniques d'obfuscation statique.

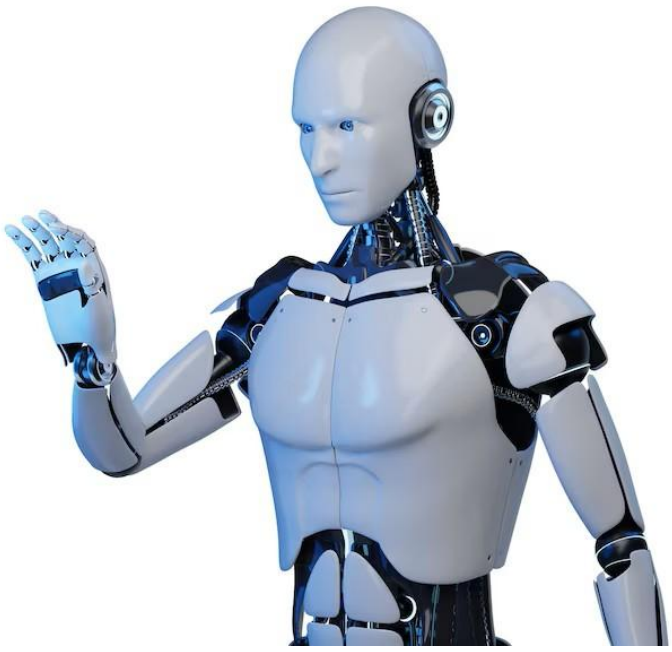
Enfin, l'analyse critique des méthodes de détection classiques, telles que les antivirus basés sur signatures et les systèmes à règles fixes, a démontré par l'argumentation et les données empiriques leurs limitations structurelles face aux malwares polymorphes, métamorphiques, aux techniques d'évasion de sandbox et aux APT mobiles. Ces limites fondamentales ne sont pas des défaillances d'implémentation corrigeables, mais des contraintes architecturales inhérentes à ces approches. Elles motivent et justifient rigoureusement le choix d'une approche par **apprentissage non supervisé**, en particulier la combinaison d'Isolation Forest, d'Autoencodeur et de One-Class SVM, une méthode apte à détecter des menaces inédites sans connaissance préalable de leur signature.

Le Chapitre 2 présentera un état de l'art détaillé des travaux académiques appliquant l'apprentissage automatique à la détection de malwares mobiles, permettant de situer notre contribution dans le paysage de la recherche existante.

Chapitre

02

INTELLIGENCE ARTIFICIELLE ET DETECTION D'ANOMALIES



2.1 INTRODUCTION :

L'intelligence artificielle (IA) constitue une branche de l'informatique dédiée au développement de systèmes automatisés capables d'imiter, voire de reproduire, des fonctions cognitives humaines. Ces fonctions englobent notamment l'apprentissage automatique, la logique de raisonnement, la prise de décision, ainsi que la perception de l'environnement ambiant. L'objectif fondamental de l'IA est d'automatiser l'exécution de tâches généralement accomplies par l'humain, à travers l'analyse de données massives, l'application d'algorithmes sophistiqués, et la modélisation prédictive.

L'émergence de l'IA remonte aux années 1950, grâce aux travaux pionniers de chercheurs tels qu'Alan Turing et John McCarthy. Depuis cette période, la discipline a connu plusieurs phases de croissance, alternant entre périodes d'avancées significatives et phases de stagnation. Depuis le début des années 2010, un essor accéléré a été observé, essentiellement dû à la disponibilité accrue de grands ensembles de données (big data), à la montée en puissance des unités de traitement graphique (GPU) et aux progrès substantiels dans le domaine des réseaux de neurones profonds (Deep Neural Networks).[23][24]

L'IA moderne se subdivise en plusieurs sous-domaines stratégiques. Le Machine Learning (apprentissage automatique) permet aux systèmes de généraliser à partir d'échantillons de données sans nécessiter de programmation explicite pour chaque tâche. Le Deep Learning exploite des architectures de réseaux de neurones multi-couches pour traiter des données complexes et non structurées. Par ailleurs, des domaines spécialisés tels que le traitement automatique du langage naturel (TALN), la vision par ordinateur, et la robotique jouent un rôle crucial dans l'évolution des capacités de l'IA.[24][16]

En termes de classification, l'IA peut être répartie en trois catégories principales, selon ses capacités fonctionnelles. L'IA faible ou étroite (Weak AI) se limite à l'exécution de tâches spécifiques, sans conscience ni compréhension véritable. L'IA forte ou générale (Strong AI) se caractérise par une capacité de raisonnement, d'apprentissage et d'adaptation comparable à celle de l'être humain, mais reste actuellement au stade expérimental ou théorique. Enfin, la Super intelligence artificielle (Super intelligence) désigne une hypothèse futuriste d'entité dotée d'intelligence surpassant de loin celle de l'humain dans tous les domaines cognitifs.

Le processus opérationnel de l'IA s'appuie sur plusieurs étapes clés. Il débute par la collecte de données pertinentes puis leur prétraitement. Celui-ci est suivi par l'analyse à l'aide

d'algorithmes de machine learning, permettant la modélisation et le paramétrage. Ensuite, le système bénéficie d'un entraînement à partir de ces modèles, avec une optimisation continue des performances au moyen de techniques d'expérimentation et d'intégration de nouveaux ensembles de données. L'IA demeure ainsi un domaine en perpétuelle évolution, dont l'objectif est d'améliorer l'efficacité des machines afin de renforcer leur utilité et leur intégration dans notre quotidien.

2.2 LE MACHINE LEARNING (APPRENTISSAGE AUTOMATIQUE) :

2.2.1 Définition :

Le Machine Learning (apprentissage automatique) constitue une sous-discipline de l'intelligence artificielle axée sur le développement de systèmes informatiques capables d'optimiser automatiquement leur performance en extrayant des motifs, structures et relations statistiques à partir de jeux de données brutes, sans intervention humaine explicite pour chaque nouvelle tâche. Ces systèmes exploitent des modèles statistiques ou algébriques pour ajuster leurs paramètres de manière itérative au cours de processus d'entraînement, permettant ainsi une généralisation à des instances inédites.

Contrairement aux paradigmes de programmation impérative traditionnels, qui reposent sur l'écriture explicite de règles décisionnelles, les algorithmes de Machine Learning identifient automatiquement des probabilités, corrélations et structures latentes inhérentes aux données d'entraînement. Les modèles construits à partir de ces analyses sont ensuite déployés pour effectuer des inférences efficaces sur de nouveaux échantillons, non présents lors de la phase d'apprentissage, en assurant une capacité d'adaptation et de généralisation optimales.

Selon Tom M. Mitchell, un programme informatique est considéré comme capable d'apprendre à partir d'une expérience lorsque, lors de l'exécution d'un ensemble défini de tâches, sa performance métrique est améliorée de manière significative en fonction de cette expérience. Cette définition met en exergue trois éléments fondamentaux : l'expérience, la tâche spécifique à accomplir, et la métrique d'évaluation de la performance. L'expérience est modélisée par un ensemble de données d'apprentissage, la tâche peut être formulée comme une opération de prédiction ou de classification, et la performance est quantifiée à l'aide de métriques telles que la précision (accuracy), le rappel (recall), ou d'autres indicateurs pertinents.[23]

Un système d'apprentissage automatique réceptionne en entrée un ensemble de données d'entraînement, appelé aussi ensemble d'apprentissage (training set), et génère un modèle de

nature mathématique, généralement une fonction paramétrée, qui établit une relation entre les variables d'entrée (features) et les cibles de sortie (targets). L'objectif principal consiste à optimiser les paramètres du modèle afin de minimiser une fonction de perte (loss function), souvent réalisée par des méthodes d'optimisation itératives telles que la descente de gradient (gradient descent). Post-entraînement, le modèle est capable de produire des prédictions sur de nouvelles données, tout en assurant une certaine capacité de généralisation. La formalisation mathématique sous-jacente confère au modèle la capacité de prendre des décisions éclairées ou de faire des inférences à partir des données auxquelles il a été exposé lors de la phase d'apprentissage [23][24]. La figure 2.1 représente les étapes successives du fonctionnement d'un algorithme de machine learning, depuis la collecte et le prétraitement des données jusqu'à l'entraînement, l'évaluation et l'amélioration itérative du modèle

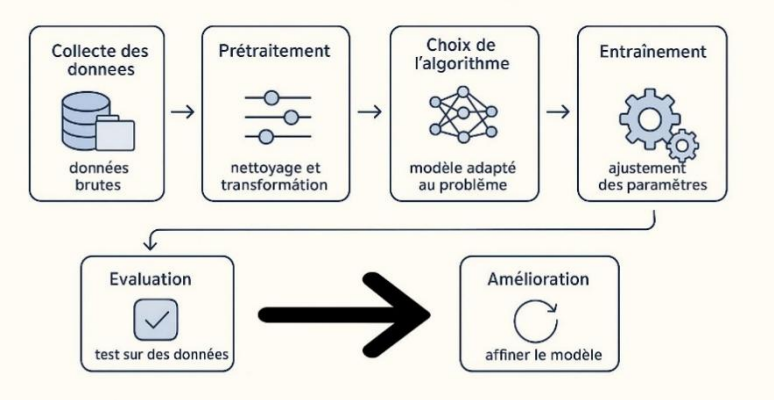


Figure 2. 1: Le fonctionnement d'un algorithme de machine learning (Généré par intelligence artificielles,2026)

2.2.2 Types d'apprentissage automatique :

Le Machine Learning se décline en trois grands paradigmes, chacun adapté à un type de problème spécifique et à la nature des données disponibles. La distinction entre ces paradigmes est essentielle pour comprendre les choix méthodologiques opérés dans ce travail. [23][16]

2.2.2.1 L'apprentissage supervisé :

La figure 2.2 représente le principe de l'apprentissage supervisé qui constitue le paradigme prédominant en apprentissage automatique, reposant sur l'utilisation d'un jeu de données annoté. Chaque instance de ce jeu de données est associée à une étiquette ou réponse cible, permettant à l'algorithme d'apprendre une fonction de prédiction à partir de ces exemples. L'objectif principal est d'apprendre une fonction approximative capable de généraliser et de prédire avec précision la réponse correcte pour des données inédites.[23][24]

Les principales catégories de tâches en apprentissage supervisé sont les suivantes :

1. Classification : La tâche consiste à attribuer une étiquette catégorielle à une nouvelle instance. Par exemple, classifier une application comme étant fiable ou non. Les algorithmes couramment employés incluent les arbres de décision, la régression logistique, et les machines à vecteurs de support (SVM).[23]

2. Régression : La tâche vise à prédire une variable continue pour une nouvelle instance. Par exemple, estimer le niveau de risque associé à une application. Les modèles fréquemment utilisés comprennent la régression linéaire et la régression polynomiale.[23]

Cependant, l'apprentissage supervisé présente des limitations notables : il exige la disponibilité de volumes substantiels de données étiquetées, ce qui peut engendrer des coûts en temps et en ressources. Par ailleurs, sa capacité à détecter de nouveaux types de menaces ou de comportements non présents lors de la phase d'entraînement reste limitée, ce qui peut compromettre la détection de menaces émergentes ou inconnues.[16][25]

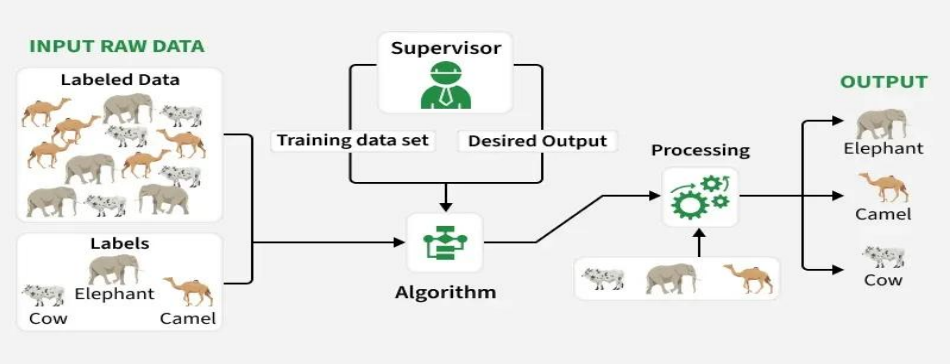


Figure 2. 2: Apprentissage Supervisé. (GeeksforGeeks (Types of Machine learning))

2.2.2.2 L'apprentissage non supervisé :

La figure 2.4 illustre le principe de l'apprentissage non supervisé qui constitue un paradigme d'analyse de données s'effectuant à partir d'un ensemble de données non étiqueté, dépourvu de réponses ou d'annotations préalablement définies. En l'absence de supervision, les algorithmes doivent déduire intrinsèquement la structure latente sous-jacente aux données, en explorant leur distribution et en identifiant des patterns auto-organisés. Cette approche repose sur une démarche exploratoire visant à extraire des regroupements naturels, des relations intrinsèques ou des anomalies au sein des données.[23][16]

Les principales applications de l'apprentissage non supervisé incluent :

- **La segmentation ou clustering** : consistant en la partition de l'espace de données en regroupements homogènes sans connaissance préalable des catégories. Les méthodes couramment employées comprennent l'algorithme K-Means, DBSCAN, ainsi que le clustering

hiérarchique agglomératif.[23]. La figure 2.3 illustre un exemple de clustering et de détection d'anomalies par la méthode KMeans, où les points isolés en croix représentent les outliers détectés selon leur distance au centre du cluster.

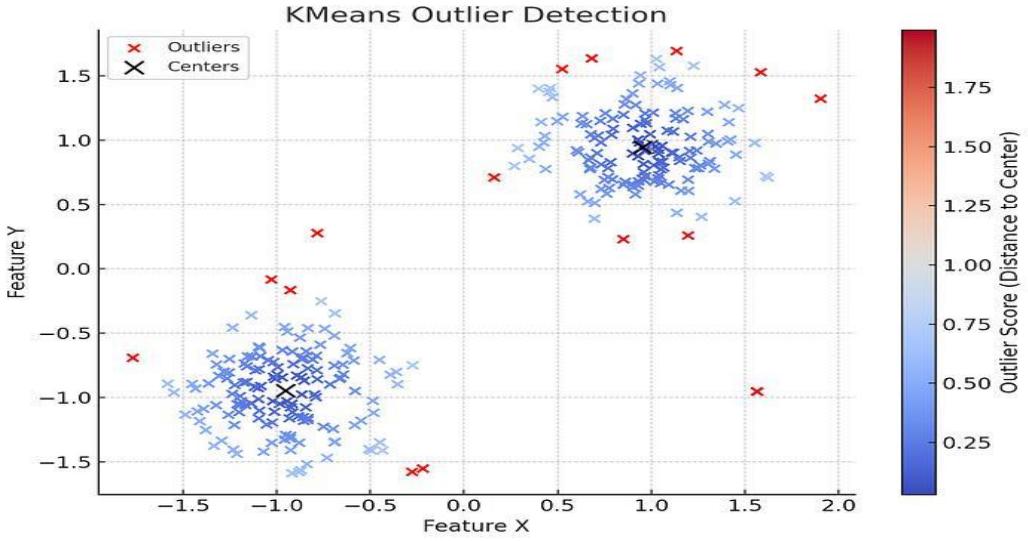


Figure 2. 3: Exemple de clustering et détection d'anomalies (Sckit-learn (KMeans Documentation))

- **La réduction de dimensionalité :** qui vise à projeter les données dans un espace de dimension inférieure tout en conservant les caractéristiques essentielles de leur structure. Les techniques associées englobent l'Analyse en Composantes Principales (ACP), t-SNE, UMAP, ainsi que les Autoencodeurs.[24]
- **La détection d'anomalies** : où l'objectif est d'identifier des instances présentant des écarts significatifs par rapport à la distribution modélisée. La détection d'anomalies constitue un élément clé dans les systèmes de sécurité.[16][26]

Dans le contexte de la sécurité mobile, l'apprentissage non supervisé offre une capacité de modélisation du comportement normal des applications légitimes. Il permet la détection en temps réel de comportements anormaux, qu'il s'agisse de malwares connus ou de menaces nouvelles et inconnues, en facilitant une approche adaptative et réactive face aux menaces émergentes.[16][27]

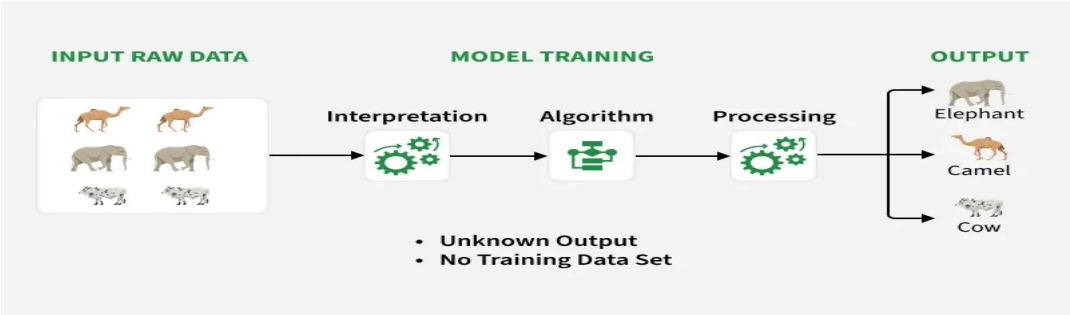


Figure 2. 4: Apprentissage Non Supervisé (GeeksforGeeks (Types of Machine learning))

2.2.2.3 L'apprentissage par renforcement :

La figure 2.5 illustre le cycle d'interaction de l'apprentissage par renforcement, où constitue une branche de l'intelligence artificielle où un agent adaptatif optimise ses stratégies de prise de décision en fonction des dynamiques de son environnement. Cet agent reçoit des retours comportementaux sous forme de récompenses ou de pénalités, quantifiés par des fonctions de récompense, en réponse à ses actions. L'objectif ultime est de maximiser la somme cumulée de ces récompenses sur une horizon temporel prolongé, par le biais de stratégies de politique optimales.

Ce paradigme est couramment implémenté dans des systèmes informatiques complexes, tels que AlphaGo, ainsi que dans la robotique autonome, pour l'apprentissage de tâches séquencées. Toutefois, cette méthodologie n'est pas appliquée dans le contexte présent, où l'étude porte sur un corpus de données statiques relatives à des comportements, sans interaction en temps réel avec un environnement dynamique.[24]

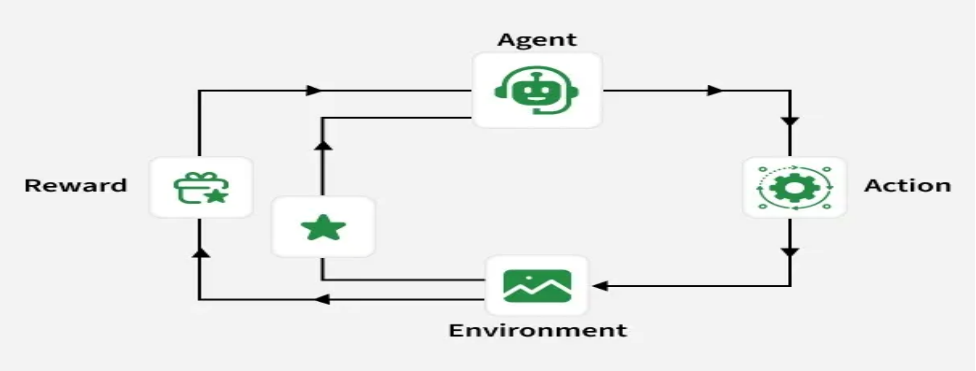


Figure 2. 5: Apprentissage par Renforcement (GeeksforGeeks (Types of Machine learning))

2.2.2.4 Apprentissage semi supervisé :

La figure 2.6 illustre le principe de l'apprentissage semi supervisé, cette approche méthodologique exploite simultanément un ensemble restreint de données étiquetées et un volume substantiel de données non étiquetées. Elle est particulièrement adaptée dans les contextes où le processus d'annotation est coûteux ou chronophage. En intégrant ces deux types de données, elle optimise la performance des modèles d'apprentissage automatique en exploitant la structure intrinsèque des données non étiquetées pour orienter et renforcer le processus d'apprentissage supervisé semi-supervisé.

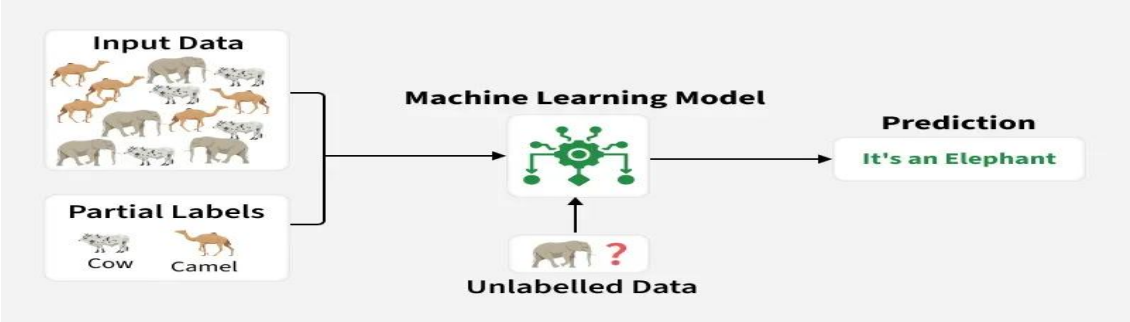


Figure 2. 6:Apprentissage Semi Supervisé (GeeksforGeeks (Types of Machine learning))

2.2.3 Justification du choix de l'apprentissage non supervisé :

Dans le contexte de la détection de logiciels malveillants sur la plateforme Android en exploitant le corpus CICMalDroid 2020, l'application de techniques d'apprentissage automatique non supervisé s'avère particulièrement pertinente pour les deux stratégies étudiées.[16][27]

Voici une analyse détaillée des bénéfices associés à l'apprentissage non supervisé :

- **Absence de besoin d'annotations exhaustives** : cette approche ne requiert pas l'étiquetage systématique de l'ensemble des données, ce qui est particulièrement crucial en présence de menaces de type « zero-day », dont les caractéristiques ne peuvent être préalablement identifiées lors de la phase d'entraînement. Le modèle apprend à modéliser le comportement normal du système, permettant ainsi la détection de toute anomalie ou déviation, indépendamment de la nature de la menace.[16][27]
- **Approche par détection d'anomalies** : cette méthodologie permet d'identifier les logiciels malveillants sans disposer de jeux de données étiquetés représentatifs de comportements malicieux. En se basant uniquement sur des données bénignes pour

l'apprentissage, elle évite les biais introduits par la présence ou l'absence d'échantillons malveillants dans le corpus d'entraînement, favorisant une détection plus généraliste.[28][26]

- **Capacité à identifier des menaces inconnues** : contrairement aux systèmes basés sur des signatures, les modèles non supervisés de détection d'anomalies sont capables de repérer des comportements malveillants inédits ou émergents. Cette aptitude constitue un avantage stratégique pour une détection proactive face à l'évolution constante des vecteurs d'attaque, renforçant ainsi la résilience du système de sécurité.[25]

2.3 LE DEEP LEARNING (APPRENTISSAGE PROFOND) :

Le Deep Learning constitue une branche spécialisée du Machine Learning, axée sur la conception et l'entraînement de réseaux de neurones artificiels à architecture profonde, caractérisés par un nombre élevé de couches cachées. Le terme « deep » indique la profondeur substantielle de ces architectures, permettant une modélisation hiérarchique de représentations de plus en plus abstraites. La profondeur du réseau, c'est-à-dire le nombre de couches, accroît sa capacité à modéliser des structures complexes et des relations non linéaires au sein des données.

Chaque niveau de ces réseaux depthwise extrait des caractéristiques de granularité croissante: les premières couches détectent des éléments rudimentaires tels que des bords, des textures ou des sons bruts. Les couches suivantes combinent ces éléments pour former des représentations plus élaborées, telles que des motifs faciaux, des mots ou des concepts abstraits. Le Deep Learning permet l'apprentissage automatique de ces caractéristiques hiérarchiques, éliminant ainsi le besoin d'un traitement manuel élaboré, souvent coûteux en temps et en ressources.[24][29]

Parmi les architectures de Deep Learning, on distingue notamment les réseaux convolutionnels pour le traitement d'images, les réseaux récurrents pour l'analyse de séquences temporelles ou textuelles, et d'autres modèles tels que les autoencodeurs, utilisés pour la compression de données, la détection d'anomalies ou l'apprentissage non supervisé. La présente étude focalise principalement sur l'application des autoencodeurs dans ce contexte.[24][29][30]

2.4 MACHINE LEARNING VS DEEP LEARNING:

Bien que le Deep Learning constitue une sous-discipline de l'apprentissage automatique, il se distingue par plusieurs caractéristiques fondamentales qui orientent leurs domaines d'application respectifs.

Les différences majeures entre l'apprentissage automatique traditionnel et le Deep Learning s'articulent autour des aspects suivants :

Ingénierie des caractéristiques : Dans le cadre de l'apprentissage automatique conventionnel, la sélection, l'extraction et la construction de caractéristiques pertinentes sont effectuées manuellement, généralement par des experts en la matière, en utilisant des techniques telles que la sélection de variables, la transformation de features ou l'ingénierie de représentation. En revanche, le Deep Learning automatise cette étape en apprenant directement des représentations hiérarchisées et abstraites à partir des données brutes, grâce à des architectures de réseaux de neurones profonds.

Volume de données requis : Les algorithmes d'apprentissage automatique conventionnels sont généralement efficaces sur des jeux de données de taille modérée, souvent de quelques milliers à quelques dizaines de milliers d'exemples. Les modèles de Deep Learning, en revanche, nécessitent des volumes de données substantiellement plus importants, souvent de l'ordre de plusieurs centaines de milliers à des millions d'échantillons, afin de converger et d'éviter le surapprentissage (overfitting) en exploitant leur capacité à modéliser des relations complexes.

Ressources computationnelles : La mise en œuvre de méthodes d'apprentissage automatique classiques est généralement compatible avec des ressources matérielles standards, telles que des processeurs centraux (CPU) avec une mémoire limitée. À l'opposé, les algorithmes de Deep Learning requièrent des ressources computationnelles accrues, notamment des unités de traitement graphique (GPU) ou des accélérateurs spécialisés, ainsi qu'une mémoire vive importante pour traiter efficacement des modèles de grande taille et des volumes de données massifs.[24]

Interprétabilité : Les modèles classiques d'apprentissage automatique, tels que les arbres de décision, les machines à vecteurs de support (SVM), ou les régressions linéaires, offrent une transparence et une interprétabilité relativement aisées, facilitant leur compréhension et leur validation par des experts. En revanche, les réseaux de neurones profonds sont souvent considérés comme des « boîtes noires » en raison de leur complexité structurale et de la difficulté à interpréter leurs représentations internes, ce qui limite leur transparence.

Capacité à modéliser les relations non linéaires : Le Deep Learning excelle dans la modélisation de relations non linéaires complexes et hiérarchiques, une aptitude essentielle pour traiter des données comportant des relations riches, hiérarchisées ou de haute dimension, telles que les images, le texte ou les signaux sensoriels. Cette capacité lui confère une efficacité supérieure dans l'apprentissage de caractéristiques discriminantes dans des espaces de

représentation non linéaires [27]. La figure 2.7 représente la comparaison entre le Machine Learning et le Deep Learning.

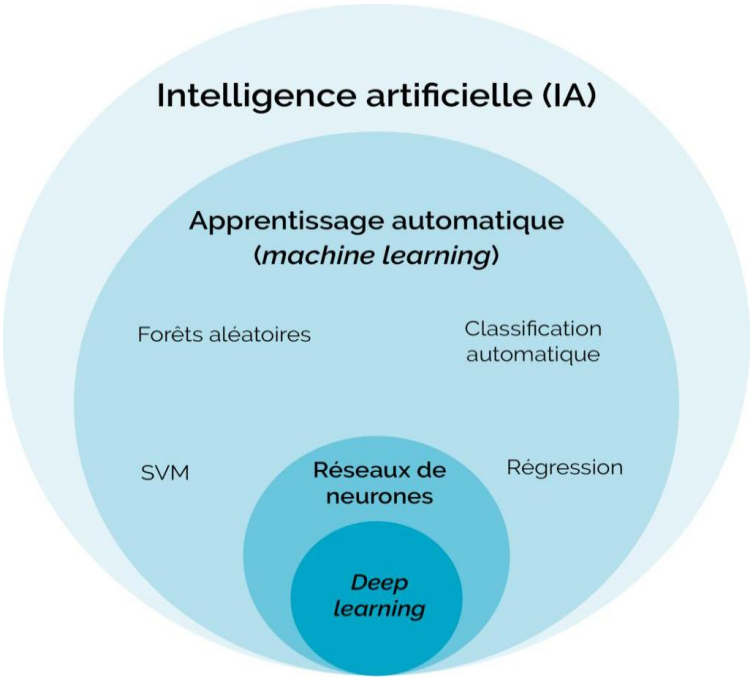


Figure 2. 7:Comparaison entre Machine Learning et Deep Learning. (Généré par intelligence artificielle ,2026)

2.5 DETECTION D'ANOMALIES PAR MACHINE LEARNING :

2.5.1 Principe générale :

La détection d'anomalies constitue un processus analytique visant à identifier des observations ou événements présentant une divergence significative par rapport au comportement attendu ou à la distribution statistique des données. Une anomalie correspond à une instance qui ne respecte pas le modèle génératif sous-jacent à l'ensemble des données.[16]

Selon Chandola, Banerjee et Kumar (2009), la figure 2.8 représente les trois types de détection d'anomalies :

1. Anomalies ponctuelles : observations isolées qui s'écartent nettement du modèle statistique global. Par exemple, un retrait bancaire exceptionnellement élevé, dérogeant aux habitudes transactionnelles de l'utilisateur.

2. Anomalies contextuelles : observations anormales dans un contexte spécifique mais non dans un cadre général. Par exemple, une requête d'accès aux contacts effectuée à 3 heures du

matin, alors que cette opération serait considérée comme normale en période de la journée habituelle.

3. Anomalies collectives : regroupements de données qui, pris ensemble, constituent un comportement déviant, alors que chaque valeur individuelle pourrait sembler conforme. Par exemple, une séquence inhabituelle d'appels système pouvant indiquer une tentative de cyberattaque.[16]

Les techniques de détection d'anomalies par apprentissage automatique opèrent en modélisant le comportement normal à partir d'un ensemble de données d'entraînement. Ensuite, chaque nouvelle instance est évaluée à l'aide d'un score d'anomalie quantifiant son degré de divergence par rapport au modèle appris. Un seuil explicite est fixé afin de classifier la donnée comme normale ou anormale en fonction de ce score.[26][16]

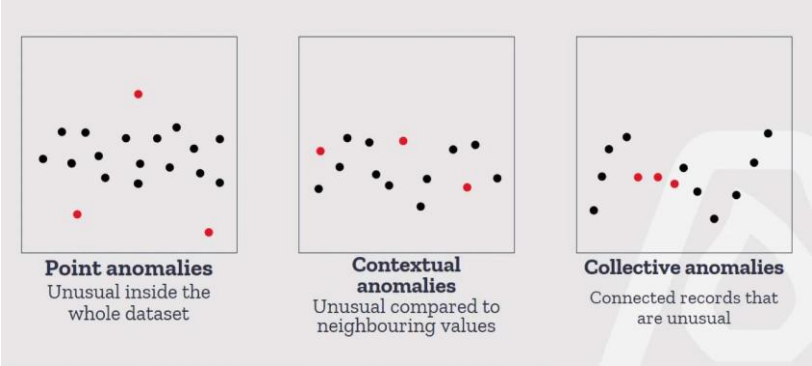


Figure 2. 8: Types de détection d'anomalies. (Généré par intelligence artificielles ,2026)

2.5.2 Approches de détection d'anomalies :

Plusieurs familles d'approches existent pour la détection d'anomalies dans le cadre de l'apprentissage automatique, chacune exploitant des principes mathématiques et statistiques spécifiques :[16]

- **Les méthodes statistiques :** modélisent explicitement la distribution probabiliste des données considérées comme normales, telles que la loi normale multivariée ou les modèles de mélanges gaussiens (GMM). La détection d'anomalies s'appuie sur le calcul d'un score basé sur la vraisemblance de chaque point sous le modèle. Néanmoins, ces méthodes souffrent de limitations dues à des hypothèses distributionnelles strictes, qui peuvent ne pas être adaptées à des données complexes ou non paramétriques.[16]
- **Les approches fondées sur la distance** : telles que k-Plus Proches Voisins (k-NN) ou Local Outlier Factor (LOF), déterminent la nature anomalies en évaluant la distance entre un point et ses voisins locaux. Les points considérés comme anormaux sont ceux dont la distance aux voisins est significativement plus grande que la moyenne locale. Cependant, l'efficacité de

ces méthodes diminue en espace de haute dimension en raison du phénomène de la « malédiction de la dimensionnalité », qui rapproche la distance entre tous les points.[25]

- **Les méthodes basées sur les modèles d'arbres** : comme la technique Isolation Forest, exploitent la simplicité de l'isolation d'anomalies via des arbres de décision aléatoires. Ces algorithmes présentent une grande scalabilité et une robustesse accrue face à la croissance de la dimensionnalité, en isolant rapidement les instances rares ou atypiques.[28]
- **Les approches à noyaux, notamment le One-Class :** Support Vector Machine (OC-SVM), recherchent une frontière de décision dans un espace de caractéristiques transformé via une fonction noyau (kernel). La méthode consiste à séparer la majorité des données normales du reste dans un espace à dimension potentiellement infinie, permettant de modéliser des distributions complexes non linéaires avec une forte capacité décisionnelle.[31]
- **Les méthodes par reconstruction** : telles que les Autoencodeurs, entraînent un modèle qui apprend à reproduire fidèlement les données normales. La qualité de reconstruction est utilisée comme indicateur d'anomalie : les instances présentant une erreur de reconstruction élevée sont considérées comme anormales, ce qui permet de détecter des déviations par rapport au comportement appris.[24]

2.5.3 Avantage de l'IA pour la détection de malwares Android :

L'utilisation de l'intelligence artificielle pour la détection d'anomalies dans les applications mobiles repose sur des techniques avancées d'apprentissage automatique, offrant plusieurs avantages par rapport aux méthodes traditionnelles basées sur l'analyse de signatures, telles que celles implémentées par les logiciels antivirus traditionnels.

Les bénéfices principaux de cette approche sont les suivants :[30]

- **Scalabilité et traitement à grande échelle**

Les modèles d'apprentissage automatique sont capables d'analyser efficacement de vastes ensembles de données, permettant le traitement simultané d'un grand nombre d'applications. Cette caractéristique est essentielle pour la détection rapide d'anomalies dans des environnements contenant des milliers d'applications, une opération qui serait prohibitive en termes de temps et de ressources si effectuée manuellement. À titre d'illustration, le dataset CICMalDroid 2020, comprenant plusieurs milliers d'applications annotées et comportant 470 métriques comportementales, met en évidence la nécessité et l'efficacité de telles techniques en contexte réel.

- **Détection d'anomalies et de menaces inconnues**

Les systèmes basés sur l'analyse d'anomalies utilisent des modèles de détection non supervisés ou semi-supervisés, capables d'identifier des comportements suspects ou non conformes aux profils normaux, sans dépendre de signatures préalablement codifiées. Cela leur confère la capacité de détecter des menaces nouvelles ou évolutives, en repérant toute déviation significative du comportement standard d'une application bénigne ou légitime.

- **Analyse multidimensionnelle et intégrée**

Les algorithmes d'apprentissage peuvent exploiter simultanément un grand nombre de caractéristiques comportementales dynamiques, telles que les appels API, les accès réseau, la consommation de ressources système (CPU, mémoire), ou encore l'accès au système de fichiers. Cette approche multidimensionnelle offre une vue d'ensemble intégrée, permettant de modéliser le comportement global de l'application de manière précise et exhaustive.

- **Adaptabilité et évolution continue**

Les modèles d'IA peuvent être périodiquement réentraînés sur de nouvelles données collectées, leur conférant une capacité d'adaptation dynamique face à l'évolution constante du paysage des menaces. Contrairement aux systèmes basés uniquement sur des règles statiques ou des signatures, cette flexibilité permet une mise à jour automatique et efficace, assurant une détection optimale face aux menaces émergentes sans intervention humaine régulière pour la modification des règles ou signatures.

2.6 ALGORITHMES DE DETECTION DANOMALIES ETUDIES :

2.6.1 Isolation Forest :

2.6.1.1 Définition :

L'algorithme Isolation Forest a été initialement proposé par Fei Tony Liu, Kai Ming Ting et Zhi-Hua Zhou lors de la 8e IEEE International Conference on Data Mining, tenue à Pise, en Italie, en 2008. Cette méthode a été ultérieurement formalise et diffusée dans un article publié dans ACM Transactions on Knowledge Discovery from Data en 2012. Conçu dans le but de pallier aux limitations des techniques classiques de détection d'anomalies, telles que celles basées sur la distance ou la densité locale, l'Isolation Forest constitue une approche innovante.[28][25]

La figure 2.9 représente le principe conceptuel de l'Isolation Forest repose sur la caractéristique que les anomalies présentent généralement une rareté et une dissimilarité vis-à-vis des observations normales. Plus précisément, ces anomalies se distinguent par leur faible fréquence et leur éloignement du centre de la distribution des données. En conséquence, dans

un arbre de décision construit aléatoirement, ces points atypiques sont plus facilement isolés que les observations conformes, qui tendent à être très similaires et regroupées en zones denses. Contrairement aux méthodes nécessitant une modélisation explicite du comportement normal (telles que les modèles basés sur la densité ou la distance), l'Isolation Forest adopte une stratégie indirecte en isolant directement les anomalies par des divisions aléatoires, permettant ainsi une détection efficace et scalable.[28]

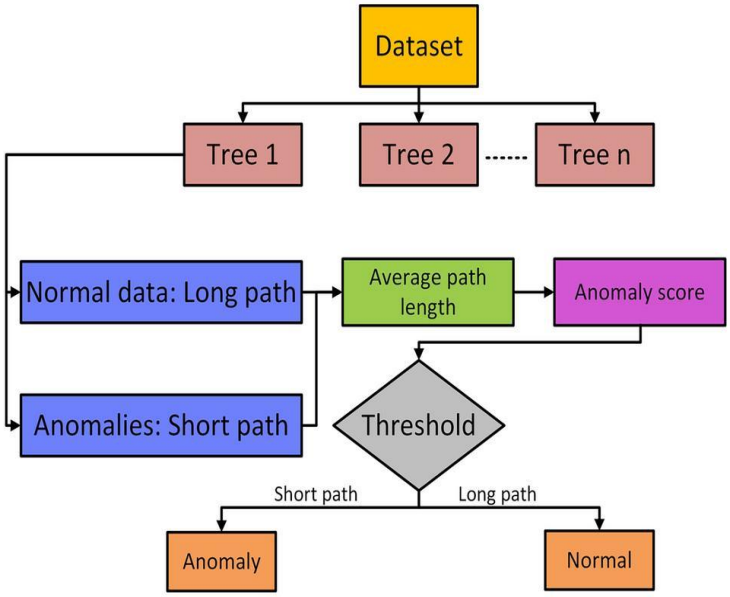


Figure 2. 9: Principe de l'algorithme Isolation Forest.(Article : Resilient cybersecurity in smart grid ICS communication using BLAKE3-driven dynamic key rotation and intrusion detection)

2.6.1.2 L'arbre d'isolation (iTree) :

La méthode de l'Isolation Forest repose sur la construction d'arbres d'isolation, désignés sous le nom d'iTree. Un iTree constitue un arbre binaire construit de manière entièrement aléatoire, selon un algorithme déterminé. Cet algorithme s'applique à un sous-échantillon de données, noté X', sélectionné aléatoirement parmi l'ensemble des données d'entraînement.[28]

Les étapes de génération d'un iTree sont les suivantes :

1. Sélection aléatoire d'une dimension (attribut) parmi les dimensions de l'espace des caractéristiques.

2. Sélection aléatoire d'une valeur de coupure dans l'étendue des valeurs prises par la dimension sélectionnée.

3. Création d'un nœud interne intégrant une règle de partitionnement basée sur la dimension choisie et la valeur de coupure déterminée. Les instances vérifiant la condition de coupure sont assignées au sous-arbre gauche, tandis que les autres le sont au sous-arbre droit.

4. Répétition récursive de ce processus sur chacun des sous-ensembles résultants, jusqu'à la satisfaction d'une condition d'arrêt. Un tel arrêt intervient typiquement lorsque chaque feuille ne contient qu'une seule instance ou lorsque la profondeur maximale de l'arbre est atteinte.

La profondeur maximale de l'arbre est contrainte de manière délibérée afin d'éviter une croissance excessive. Il est à noter que les anomalies sont généralement détectées bien avant d'atteindre cette limite de profondeur. Cette restriction permet également de diminuer l'utilisation mémoire et le temps de traitement nécessaires à la construction de l'arbre.[28]

2.6.1.3 La zone d'isolation forestière et l'évaluation du score d'anomalie :

Une forêt d'isolation (Isolation Forest) consiste en un ensemble de t arbres d'isolement (t iTree), chacun construit de manière indépendante à partir de sous-échantillons distincts de taille ψ. Pour évaluer le score d'anomalie d'une instance x, celle-ci est hiérarchiquement traversée par chaque arbre, où la mesure clé est la longueur du chemin h(x), correspondant au nombre de coupures traversées depuis la racine jusqu'à la feuille terminale. Cette longueur de chemin est ajustée pour refléter le fait que l'arrêt de l'arbre peut intervenir avant la profondeur maximale, en tenant compte de la profondeur atteinte.[28]

Le score d'anomalie global s(x, ψ) est défini par la formule suivante, qui assure une normalisation par la longueur moyenne de chemin théorique c(ψ) dans un arbre de recherche binaire construit sur un échantillon de taille ψ. L'équation (1) définit le score d'anomalie s(x, ψ), calculé comme une puissance de 2 normalisée par la longueur moyenne de chemin théorique

$$s(x, \psi) = 2^{-\frac{E[h(x)]}{c(\psi)}} \qquad \text{[eq (1)]}$$

Où E[h(x)] désigne la moyenne empirique des longueurs de chemin h(x) obtenues à travers les t arbres. L'équation 2 représente la longueur de chemin moyenne c(ψ) est donnée par l'expression suivante, faisant intervenir le nombre harmonique H(i) et la constante d'Euler-Mascheroni γ :

$$c(\psi) = 2H(\psi - 1) - \left(\frac{2(\psi-1)}{\psi}\right) \qquad \text{[eq (2)]}$$

Avec H(i) désignant le i-ième nombre harmonique, défini par H(i) = ln(i) + γ où γ ≈ 0,5772 est la constante d'Euler-Mascheroni. Cette normalisation garantit que la valeur du score s(x, ψ) est bornée dans l'intervalle (0, 1).[28]

L'interprétation du score s(x, ψ) est la suivante :[28]

- Pour s(x, ψ) proche de 1 : la longueur de chemin moyenne est très courte, indiquant que l'instance est aisément isolée, et par conséquent très probablement anomalie.

- Pour s(x, ψ) proche de 0,5 : la longueur de chemin est approximativement égale à c(ψ), ce qui indique que l'instance occupe une position normale dans l'espace de caractéristiques.

- Pour s(x, ψ) proche de 0 : la longueur de chemin est élevée, signifiant que l'instance est difficile à isoler, ce qui suggère qu'elle est vraisemblablement normale.

2.6.1.4 Hyperparamètres et configuration :

Le tableau 2.1 représente l'hyperparamètre de l'Isolation Forest. L'algorithme Isolation Forest repose sur trois hyperparamètres essentiels. La calibration de ces paramètres influence directement la performance et l'efficacité de la détection d'anomalies réalisée par l'algorithme.[28][26]

Hyperparamètre	Description	Valeur recommandé
n_estimators (t)	Nombre d'arbres iTree dans la forêt	100(stabilité élevée)
max_samples (ψ)	Taille du sous-échantillon par arbre	256(optimal)
Contamination	Proportion estimée d'anomalies	Proportion réelle du jeu test
max_features	Nb de features considérées par coupure	1.0(toutes les features)
random_state	Graine aléatoire pour reproductibilité	Fixer pour l'étude

Tableau 2. 1:Hyperparamètres de l'Isolation Forest [3,6]

2.6.1.5 Propriétés théoriques et complexité :

L'Isolation Forest possède des propriétés théoriques distinctives qui le différencient des autres méthodes d'analyse.[28]

- **La complexité temporelle pour l'entraînement** La complexité algorithmique de l'Isolation Forest est de l'ordre de $O(t \times \psi \times \log \psi)$, où t désigne le nombre d'arbres et ψ la taille de chaque sous-échantillon. Cette complexité est indépendante du nombre total d'observations n, ce qui permet à l'algorithme d'être performant sur de grands ensembles de données comprenant plusieurs millions d'instances, en assurant un traitement en temps linéaire.
- **La consommation mémoire** de l'Isolation Forest est estimée à $O(t \times \psi)$, chaque arbre nécessitant le stockage d'un maximum de $2\psi - 1$ nœuds.
- **La complexité d'inférence** c'est-à-dire le temps requis pour évaluer une seule instance, est de $O(t \times \log \psi)$, ce qui confère à l'algorithme une performance optimale, notamment dans le cadre de prédictions en temps réel
- Par ailleurs, l'Isolation Forest se caractérise par son **indépendance** vis-à-vis de la distribution des données. En effet, cet algorithme est non paramétrique et ne fait aucune hypothèse quant à la forme ou la distribution des données, contrairement aux méthodes basées sur des hypothèses gaussiennes.[28]

2.6.1.6 Avantages et limites :

L'Isolation Forest présente plusieurs avantages majeurs :[28]

- **Efficacité en haute dimension** : Contrairement aux méthodes basées sur la distance (telles que k-NN) ou la densité (comme LOF), qui souffrent du phénomène de la malédiction de la dimension, l'Isolation Forest reste performante dans des espaces à haute dimension, notamment ceux comportant jusqu'à 470 dimensions (par exemple, dans le cadre de CICMalDroid 2020).
- **Faible sensibilité au réglage des hyperparamètres :** Les paramètres par défaut ($t=100$, $\psi=256$) offrent généralement de bonnes performances sans nécessiter d'optimisation fine.
- **Robustesse face au bruit** : Le sous-échantillonnage aléatoire contribue à limiter l'impact des anomalies bruitées lors de la construction des arbres.
- **Scalabilité linéaire** : L'algorithme est adapté à l'analyse de grands volumes de données industriels.

Néanmoins, l'Isolation Forest présente également certaines limitations :[28]

- **Effet de masquage :** Lorsque les anomalies sont regroupées en clusters denses, leur détection peut devenir difficile. La variante Extended Isolation Forest (Hariri et al., 2018) atténue partiellement ce problème en utilisant des hyperplans de coupure aléatoires plutôt que des coupures axiales.

- **Dépendance à l'estimation du taux d'anomalies** : Une incorrecte estimation de la proportion d'anomalies (paramètre de contamination) peut nuire aux performances, notamment en termes de rappel.
- **Inadéquation aux données discrètes ou catégorielles** : Conçu pour des variables continues, l'algorithme nécessite un encodage préalable pour traiter des données mixtes ou discrètes.

2.6.2 Autoencodeur :

2.6.2.1 Définition :

Les autoencodeurs constituent un type de réseau de neurones qui n'exige pas de données étiquetées pour son apprentissage. Le principe remonte aux travaux de Rumelhart, Hinton et Williams (1986) sur la rétropropagation du gradient, qui ont rendu possible l'entraînement de réseaux de neurones multicouches, permettant ainsi l'émergence d'architectures encodeur-décodeur symétriques capables de comprimer puis de reconstruire des données.

En 2016, Goodfellow, Bengio et Courville ont formalisé le concept d'autoencodeurs comme des réseaux visant à reproduire leurs entrées en sortie en passant par un goulot d'étranglement, ce qui les oblige à apprendre une représentation compacte des données d'entrée.[24]

L'utilisation des autoencodeurs pour la détection d'anomalies s'est progressivement développée. Le principe repose sur le fait qu'un réseau entraîné exclusivement sur des données normales parvient à les reconstruire avec une grande précision, tandis que sa capacité à reconstruire des données anormales, qu'il n'a jamais rencontrées, reste limitée, entraînant une erreur de reconstruction plus élevée. Cette propriété constitue la base de leur application dans la détection d'anomalies.

Ainsi, les autoencodeurs disposent d'une capacité intrinsèque à identifier des données atypiques, en se concentrant sur les caractéristiques des données normales qu'ils ont appris à reconnaître.[24][16]

2.6.2.2 Architecture d'Autoencodeur :

Un Autoencodeur classique se compose de deux sous-réseaux symétriques et complémentaires : un encodeur et un décodeur.[24]

a) L'encodeur

L'encodeur est une fonction paramétrique qui transforme les données d'entrée de grande dimension en une représentation plus compacte dans un espace latent de dimension réduite. Il est composé d'une succession de couches entièrement connectées, où chaque couche applique

une transformation linéaire suivie d'une fonction d'activation de type ReLU. Au fur et à mesure de la propagation des données dans le réseau, la taille des couches diminue progressivement, permettant ainsi de compresser l'information tout en conservant les caractéristiques essentielles des données normales. [24]

b) Le goulot d'étranglement (Bottleneck)

La couche centrale, appelée goulot d'étranglement, possède la dimension la plus faible de l'architecture. Elle contient la représentation compacte des données d'entrée et constitue l'élément clé de l'autoencodeur. Une dimension trop réduite peut provoquer une perte importante d'informations et rendre la reconstruction des données normales moins précise. À l'inverse, une dimension trop élevée peut conduire au surapprentissage, en permettant au réseau de mémoriser les données d'entraînement et de reconstruire également les anomalies, ce qui diminue les performances de détection. En pratique, la dimension de l'espace latent est souvent choisie entre 5 % et 15 % de la dimension initiale des données. [24][31]

c) Le décodeur gφ

La figure 2.10 représente l'architecture d'un Autoencodeur. Le décodeur est une fonction symétrique de l'encodeur dont le rôle est de reconstruire les données d'origine à partir de leur représentation compacte dans l'espace latent. Il reçoit en entrée le vecteur latent et tente de produire une reconstruction aussi proche que possible des données initiales. Comme l'encodeur, il est constitué de plusieurs couches entièrement connectées, mais la dimension des couches augmente progressivement afin de revenir de l'espace latent réduit vers la dimension initiale des données. La couche de sortie utilise généralement une activation linéaire pour les données

réelles non bornées, ou une fonction sigmoïde lorsque les données sont binaires ou normalisées dans l'intervalle [0,1]. [24]

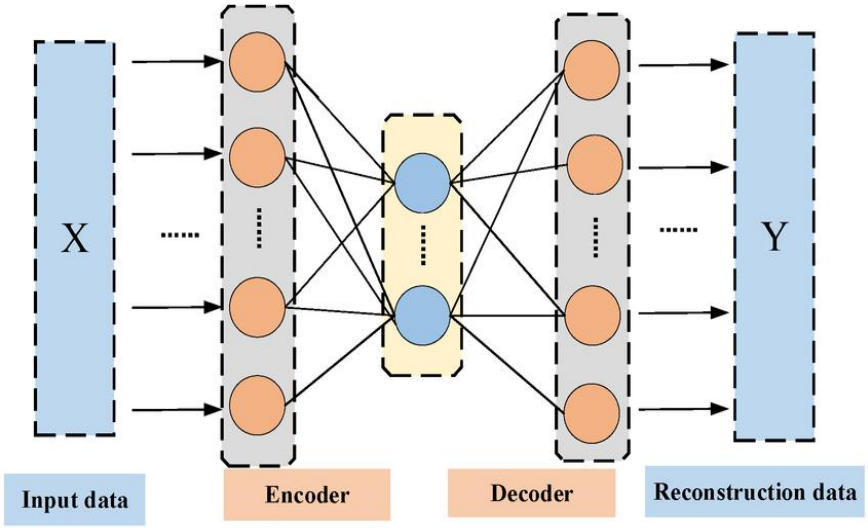


Figure 2. 10:Architecture d'un autoencodeur. (Article publié dans Energies Journal)

2.6.2.3 Entrainement et fonction de perte :

L'entraînement d'un autoencodeur consiste à ajuster les paramètres de l'encodeur et du décodeur afin de minimiser l'erreur de reconstruction sur l'ensemble des données d'apprentissage. . L'équation (3) exprime la fonction de perte MSE, calculée comme la moyenne des erreurs quadratiques entre les données d'entrée et leur reconstruction par l'autoencodeur :[24]

$$L(\theta, \varphi) = \frac{1}{n}\sum_{i=1}^{n} \left\| x_i - g_\varphi\left(f_\theta(x_i)\right)\right\|^2 \qquad \text{[eq(3)]}$$

Pour optimiser ces paramètres, on emploie généralement la méthode de la descente de gradient stochastique (SGD) ou ses variantes adaptatives. En pratique, l'algorithme Adam (Kingma & Ba 2014) est fréquemment privilégié en raison de sa rapidité de convergence et de sa capacité à gérer efficacement des données de différentes échelles. Les gradients sont calculés en rétropropagation à travers l'ensemble constitué par l'encodeur et le décodeur, permettant ainsi d'ajuster simultanément l'ensemble des paramètres.[24][31]

2.6.2.4 Détection d'anomalies par erreur de reconstruction :

À l'issue de la phase d'entraînement réalisée sur un ensemble de données saines, constitué ici des applications bénignes du jeu de données CICMalDroid 2020, le modèle est utilisé pour évaluer chaque nouvelle observation. Pour cela, les données d'entrée sont d'abord compressées

par l'encodeur puis reconstruites par le décodeur. Le modèle calcule ensuite l'erreur de reconstruction, c'est-à-dire la différence entre les données originales et leur version reconstruite. Une faible erreur indique généralement que l'observation ressemble aux données normales apprises durant l'entraînement, tandis qu'une erreur élevée peut signaler un comportement anormal ou malveillant. [24][16]

L'erreur de reconstruction obtenue est ensuite comparée à un seuil de décision. Si cette erreur dépasse le seuil fixé, l'observation est considérée comme anormale ; sinon, elle est classée comme normale. En pratique, ce seuil est déterminé à partir d'un ensemble de validation composé de données normales. Les erreurs de reconstruction sont calculées pour toutes les instances de validation, puis le seuil est choisi comme une valeur élevée de la distribution des erreurs, généralement le 95e ou le 99e percentile. Ainsi, les observations présentant une erreur supérieure à ce seuil sont identifiées comme des anomalies. [24]

2.6.2.5 Hyperparamètres clés :

Les principaux hyperparamètres à ajuster pour l'autoencodeur dans le cadre de notre étude sont les suivants [24][31] :

- **Taux d'apprentissage (η)** : il contrôle l'amplitude des mises à jour des paramètres. Avec l'algorithme Adam, une valeur de $\eta = 10^{-4}$ est généralement efficace. Un taux trop élevé peut provoquer des oscillations, tandis qu'un taux trop faible entraîne une convergence lente.
- **Nombre d'époques :** il correspond au nombre de passages complets sur l'ensemble de données d'apprentissage. La technique de early stopping permet de surveiller la perte sur le jeu de validation et d'interrompre l'entraînement lorsque celle-ci cesse de diminuer, ce qui contribue à prévenir le surapprentissage (overfitting).
- **Taille du lot (batch size) :** il indique le nombre d'instances traitées lors de chaque mise à jour. Un compromis entre 32 et 64 permet d'assurer un bon équilibre entre la vitesse de convergence et la qualité du gradient.
- **Régularisation L2 (weight decay) :** cette méthode pénalise les poids excessivement élevés afin de réduire le risque de surapprentissage.
- **Dropout :** cette technique consiste à désactiver aléatoirement, lors de chaque itération, un pourcentage p des neurones. Cela oblige le réseau à apprendre des représentations plus robustes et à ne pas dépendre de neurones spécifiques.[24]

2.6.2.6 Avantage et limites :

Avantages de l'Autoencodeur :

• **Représentation des non-linéarités complexes :** Grâce à l'usage de fonctions d'activation non linéaires, un autoencodeur peut modéliser des distributions de données aux formes très variées, surpassant ainsi les méthodes linéaires telles que l'Analyse en Composantes Principales (ACP).

• **Adaptation à des espaces de haute dimension :** L'autoencodeur s'adapte efficacement à des espaces de features de grande dimension (par exemple, 470 caractéristiques) sans subir les effets liés à la malédiction de la dimensionnalité.

• **Flexibilité architecturale :** La conception du réseau peut être adaptée en ajustant la profondeur, la largeur, ainsi que le type des couches (denses, convolutives, récurrentes) en fonction des particularités du domaine d'application.

• **Intégration dans les environnements de Deep Learning :** L'autoencodeur peut bénéficier des optimisateurs avancés (tels qu'Adam ou AdaGrad), des techniques de régularisation modernes (Dropout, Batch Normalization) et peut bénéficier de l'accélération par GPU.

Inconvénients de l'Autoencodeur : [24][16]

• **Coût computationnel important** : Entraîner un réseau profond demande des ressources matérielles et du temps considérables, largement supérieurs à ceux requis par des méthodes comme la Forêt d'Isolation.

• **Nécessité de réglages fins :** La détermination de l'architecture optimale (nombre de couches, taille du goulot d'étranglement, paramètres de régularisation) repose sur une série d'expérimentations. Des choix sous-optimaux peuvent entraîner une dégradation significative des performances.

• **Risque de surapprentissage :** Un réseau doté d'une capacité excessive (bottleneck trop large ou réseau trop profond) risque de mémoriser précisément les anomalies lors de l'apprentissage, ce qui peut réduire sa capacité à discriminer efficacement. Il est donc essentiel d'assurer une régularisation appropriée.

2.6.3 One-Class SVM (OC-SVM):

2.6.3.1 Définition:

Le One-Class SVM (OC-SVM) constitue une méthode de détection d'anomalies développée par Bernhard Schölkopf, Robert Williamson, Alexander Smola, John Shawe-Taylor et John Platt, initialement présentée dans un article publié aux Advances in Neural Information Processing Systems (NIPS) en 1999, puis approfondie dans la revue Neural Computation en 2001. Cette approche s'inscrit dans la lignée des Systèmes à Vecteurs de Support (SVM)

conceptualisés par Cortes et Vapnik en 1995, en l'adaptant spécifiquement au cadre de l'apprentissage à une seule classe, dite « one-class learning », où seules des données représentatives de la classe normale sont disponibles pour l'entraînement.[31]

L'idée fondamentale du OC-SVM consiste à identifier, dans un espace de caractéristiques de dimension élevée (l'espace de Hilbert induit par un noyau), un hyperplan optimal séparant les données d'entraînement de l'origine avec la marge la plus large possible. Les instances situées de l'autre côté de cet hyperplan, proches de l'origine, sont considérées comme potentielles anomalies, tandis que celles du même côté que l'hyperplan sont classées comme normales.[31]

2.6.3.2 Fondements théoriques des SVM :

Pour comprendre le fonctionnement d'un One-Class SVM (OC-SVM), il est essentiel de maîtriser d'abord les principes des SVM classiques. Un SVM standard cherche à déterminer un hyperplan capable de séparer de manière optimale deux classes de données tout en maximisant la marge de séparation. Cette marge représente la distance minimale entre l'hyperplan et les points d'entraînement les plus proches, appelés vecteurs de support. Plus cette marge est grande, meilleure est généralement la capacité de généralisation du modèle. [31]

L'un des principaux avantages des SVM réside dans l'utilisation du « trick du noyau ». Cette approche permet de projeter implicitement les données dans un espace de dimension élevée, voire infinie, afin de rendre les données plus facilement séparables. Au lieu de calculer explicitement cette transformation, le modèle utilise une fonction noyau qui mesure directement la similarité entre deux observations dans l'espace transformé, ce qui réduit considérablement le coût de calcul. [31]

Les noyaux les plus couramment exploités comprennent :

- **Le noyau linéaire :** simple et efficace pour les données linéairement séparables.
- **Le noyau polynomial** : capable de modéliser des relations polynomiales d'ordre d.
- **Le noyau RBF (Radial Basis Function) :** fréquemment utilisé en pratique, correspondant à un espace de caractéristiques de dimension infinie.
- **Le noyau sigmoïde :** qui présente une analogie avec les réseaux de neurones à une couche cachée.

2.6.3.3 Interprétation géométrique :

Sur le plan géométrique, le One-Class Support Vector Machine (OC-SVM) cherche à construire une frontière de décision entourant la majorité des données d'entraînement

considérées comme normales. Cette frontière prend la forme d'une région convexe, ou d'une hypersphère dans l'espace de Hilbert généré par la fonction noyau. Les observations situées à l'intérieur de cette région sont considérées comme normales, tandis que celles qui se trouvent à l'extérieur sont identifiées comme des anomalies. [31]

La figure 2.11 représente la délimitation de la région normale construite par l'OC-SVM, où les points situés à l'intérieur de l'ellipse sont considérés comme normaux, tandis que le point rouge à l'extérieur est identifié comme une anomalie. Grâce à l'utilisation des fonctions noyaux, notamment le noyau RBF, le modèle peut créer des frontières de décision non linéaires capables de s'adapter efficacement à des distributions complexes de données normales. Cette flexibilité permet au modèle de mieux représenter la structure réelle des données, même lorsque celles-ci ne sont pas séparables de manière linéaire dans l'espace d'origine. [31]

Pour chaque nouvelle observation, l'OC-SVM calcule un score d'anomalie basé sur sa position par rapport à la frontière de décision. Plus une observation est éloignée de la région représentant les données normales, plus son score d'anomalie est élevé et plus elle est susceptible d'être considérée comme anormale ou malveillante. À l'inverse, les observations proches de la frontière ou situées à l'intérieur de la région normale obtiennent des scores faibles et sont généralement classées comme normales. [31]

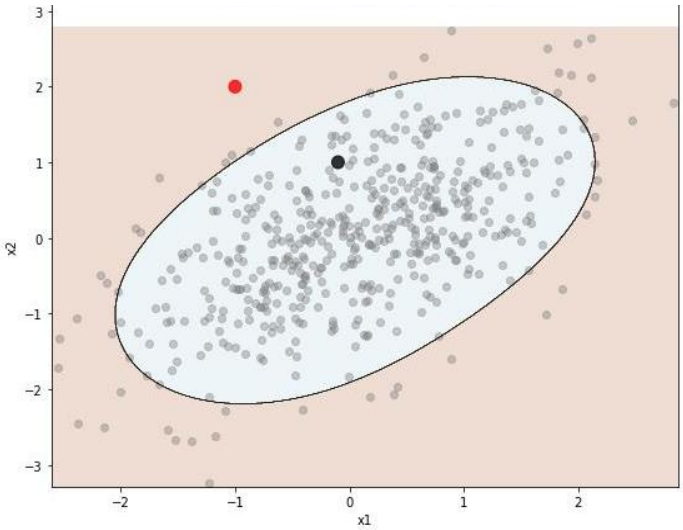


Figure 2. 11:Délimitation de la région normale par One-Class SVM (ML Blog _ One Class SVM Anomaly Detection)

2.6.3.4 Hyperparamètres et configuration :

Le tableau 2.2 présente l'hyperparamètres du One-Class SVM. L'OC-SVM comporte deux hyperparamètres principaux à ajuster, en complément des paramètres spécifiques au noyau sélectionné. [26][31]

Hyperparamètre	Rôle	Valeur typique
ν (nu)	Borne sup fraction de support vectors / borne inf fraction d'anomalies	0.05 à 0.20
Noyau (Kernel)	Type de fonction noyau utilisée	RBF (recommandé)
γ(gamma)	Paramètre de largeur du noyau RBF	'scale' ou 'auto'
Cache size	Mémoire cache pour la matrice noyau (MB)	500-1000 MB

Tableau 2. 2:Hyperparamètres du One-Class SVM

2.6.3.5 Réglage du paramètre γ du noyau RBF :

Le paramètre γ du noyau RBF contrôle l'étendue de l'influence des vecteurs de support sur la frontière de décision du modèle. Il joue un rôle essentiel dans la capacité du One-Class SVM à s'adapter à la distribution des données. Une valeur élevée de γ produit une frontière de décision très localisée, dans laquelle chaque vecteur de support influence principalement les observations proches. Cette configuration permet de modéliser des structures complexes, mais elle augmente également le risque de surapprentissage. À l'inverse, une faible valeur de γ génère une frontière plus lisse et plus générale, ce qui peut conduire à un sous-apprentissage si le modèle devient incapable de capturer correctement la structure des données.

Dans la bibliothèque scikit-learn, la valeur 'scale' est souvent utilisée par défaut pour définir automatiquement γ. Cette approche calcule γ en fonction du nombre de caractéristiques et de la variance des données, permettant ainsi une normalisation implicite adaptée à l'échelle des variables. Elle constitue généralement un bon point de départ avant d'effectuer un ajustement plus précis des hyperparamètres du modèle.[31]

2.6.3.6 Avantages et limites :

Avantages du One-Class SVM :

- **Flexibilité de la frontière de décision :** grâce à la technique du kernel, le One-Class SVM est capable d'apprendre des frontières de formes arbitrairement complexes dans l'espace d'origine, ce qui lui permet de s'adapter à des distributions de données non convexes.
- **Garanties théoriques solides** : basé sur la théorie de l'apprentissage statistique de Vapnik-Chervonenkis (VC), le One-Class SVM bénéficie de bornes de généralisation formelles.
- **Interprétabilité partielle** : les vecteurs supports identifiés par l'algorithme représentent les exemples d'entraînement les plus déterminants pour la frontière de décision, offrant ainsi une certaine interpretabilité.
- **Robustesse aux outliers d'entraînement :** le paramètre ν permet de tolérer une fraction contrôlée d'anomalies dans les données d'entraînement, rendant le modèle robuste face à un étiquetage imparfait. [31]

Limitations du One-Class SVM :

- **Complexité computationnelle quadratique :** le temps d'entraînement est en O(n^2) ou O(n^3) selon l'implémentation, ce qui peut entraîner des ralentissements importants pour de très grands jeux de données (n > 10 000).
- **Difficulté dans le réglage des hyperparamètres γ et ν** : la sélection conjointe de ces paramètres sans données étiquetées disponibles est complexe et requiert souvent des heuristiques ou une validation croisée sur les anomalies.
- **Sensibilité à l'échelle des variables :** le One-Class SVM est sensible à la normalisation des features. Une standardisation préalable (par exemple avec StandardScaler ou MinMaxScaler) est essentielle avant l'entraînement. [26][31]

2.7 COMPARAISON DES TROIS ALGORITHMES :

Le tableau 2.3 offre une synthèse comparative des trois algorithmes, en tenant compte des critères les plus pertinents pour notre étude. [28][24][31]

Critère	Isolation Forest	Autoencodeur	One-Class SVM
Famille	Arbres aléatoires	Deep Learning	Méthodes à noyaux

Complexité entrainement	O(t ψ logψ) linéaire	O(n époques) moyen	O(n^2) à O(n^3) lent
Haute dimensionnalité	Excellent	Excellent	Moyen
Non-linéarités	Oui (partitions)	Très bon (DL)	Oui (noyau RBF)
Interprétabilité	Moyenne	Faible	Partielle
Réglage hyperparamètre	Simple	Complexe	Moyen
Besoin GPU	Non	Recommandé	Non
Scalabilité n> $100K$	Excellente	Bonne	Faible

Tableau 2. 3:Comparaison des trois algorithmes de détection d'anomalies

2.8 CONCLUSION :

Ce chapitre a posé les fondements théoriques essentiels à notre étude. Il a présenté les concepts fondamentaux de l'intelligence artificielle, en distinguant les différents paradigmes du Machine Learning supervisé, non supervisé, semi-supervisé et par renforcement et en illustrant comment le Deep Learning en constitue une extension performante. Le choix de privilégier l'apprentissage non supervisé repose sur sa capacité à modéliser le comportement normal des applications sans nécessiter un étiquetage exhaustif, facilitant ainsi la détection de menaces à la fois connues et inconnues, y compris les attaques de type zero-day [23][24][16].

Trois algorithmes complémentaires ont été présentés et analysés en détail : l'Isolation Forest, qui repère les anomalies par partitionnement binaire aléatoire avec une complexité linéaire adaptée à nos 470 caractéristiques [28] ; l'Autoencodeur, qui identifie les anomalies par une erreur de reconstruction élevée, grâce à une architecture encodeur–goulot–décodeur [24] ; et le One-Class SVM, qui délimite géométriquement la région normale dans un espace à noyau RBF, bénéficiant de garanties théoriques solides [31].

Une fois ces bases théoriques établies, le chapitre suivant se tourne vers la mise en pratique. Le Chapitre 03 décrira la mise en œuvre expérimentale complète de ces trois algorithmes sur le

jeu de données CICMalDroid 2020, en détaillant le processus de prétraitement, la configuration des hyperparamètres, ainsi que l'analyse comparative des performances selon les métriques retenues précision, rappel, score F1 et AUC-ROC dans le but d'identifier l'approche la plus efficace pour la détection de malwares Android [28][24][31][27].

CHAPITRE

03

IMPLEMENTATION ET EVALUATION DU SYSTEME DE DETECTION D'ANOMALIES



3.1 INTRODUCTION :

Face à cette réalité, les approches traditionnelles de détection atteignent rapidement leurs limites. Elles reposent en général sur une connaissance préalable des menaces, ce qui les rend peu efficaces dès qu'un malware nouveau ou modifié fait son apparition. C'est pourquoi ce travail s'oriente vers une approche différente, basée sur la détection d'anomalies comportementales. Au lieu de chercher à identifier des malwares connus, l'objectif est d'apprendre ce qu'est un comportement normal, puis de signaler tout ce qui s'en écarte de manière significative. Cette logique, portée par des algorithmes d'apprentissage non supervisé et semi-supervisé, offre une flexibilité que les méthodes classiques ne peuvent pas atteindre, notamment pour faire face à des menaces encore inconnues au moment du déploiement du système.

Ce chapitre présente l'ensemble du travail réalisé, depuis la préparation des données jusqu'à l'évaluation des modèles. Nous présentons d'abord l'environnement de développement utilisé, puis le jeu de données CICMalDroid 2020 sur lequel reposent toutes les expérimentations, avec une analyse exploratoire permettant de mieux comprendre sa structure. Le pipeline de prétraitement est ensuite détaillé, suivi de la description des trois modèles implémentés : Isolation Forest, One-Class SVM et Autoencodeur. Deux approches expérimentales sont ensuite comparées : une approche one-class entraînée uniquement sur des données bénignes, et une approche semi-supervisée utilisant le dataset complet après rééquilibrage des classes. Les résultats obtenus sont analysés et discutés, puis comparés à différents travaux de la littérature afin de situer les performances du système proposé.

La démarche suivie dans ce chapitre repose sur une logique rigoureuse et méthodique, depuis le traitement des données jusqu'à l'interprétation des résultats. L'objectif est de montrer qu'un système entraîné uniquement sur des applications normales peut détecter efficacement des malwares, y compris ceux qu'il n'a jamais vus auparavant. Chaque choix technique effectué, que ce soit le modèle utilisé, la façon de préparer les données ou la manière d'évaluer les performances, est justifié et documenté afin que les résultats obtenus soient fiables et reproductibles.

3.2 ENVIRONNEMENT DE DEVELOPPEMENT :

3.2.1 Plateforme Kaggle Notebook :

L'ensemble des expérimentations menées dans cette étude a été réalisé sur la plateforme Kaggle Notebooks, un environnement de développement interactif basé sur Python et hébergé dans le cloud. Ce choix s'explique par plusieurs avantages techniques et scientifiques.

D'une part, la plateforme met gratuitement à disposition des ressources de calcul accélérées, notamment des GPU NVIDIA Tesla P100 et T4, permettant d'exécuter efficacement les phases d'entraînement et d'évaluation des modèles de deep learning. D'autre part, Kaggle assure une meilleure reproductibilité expérimentale grâce à la gestion centralisée des dépendances logicielles, des versions de bibliothèques et de l'environnement d'exécution. En outre, la possibilité de partager publiquement les notebooks favorise la transparence scientifique, la collaboration et la réutilisation des travaux par la communauté de recherche [32].

Par ailleurs, le jeu de données CICMalDroid 2020 utilisé dans ce travail a été intégré directement dans l'environnement Kaggle sous l'identifiant *raniabenyelloul/dtection-danomalies-dans-les-app*, ce qui a permis un accès rapide aux données ainsi qu'une gestion simplifiée des étapes de prétraitement, d'entraînement et de validation des modèles.

3.2.2 Langage et bibliothèques :

Le langage de programmation utilisé est Python 3.10, dont la richesse de l'écosystème scientifique ainsi que la clarté du code en font un choix particulièrement pertinent pour les applications de machine learning [33]. Le tableau 3.1 présente un résumé des principales bibliothèques utilisées ainsi que leurs versions respectives.

Bibliothèque	Version	Rôle dans le projet
Python	3.10.12	Langage de programmation principal
NumPy	1.24.3	Calcul numérique manipulation de tableaux
Pandas	2.0.1	Chargement et manipulation du dataset CSV
Scikit-learn	1.2.2	Isolation Forest, OC-SVM, StandardScaler, métriques

TensorFlow	2.12.0	Construction et entrainement de l'Autoencodeur (Keras API)
SHAP	0.42.1	Analyse d'interprétabilité (TreeExplainer)
Matplotlib	3.7.1	Visualisations : courbes ROC, matrices de confusion, histogrammes
Seaborn	0.12.2	Heatmaps de corrélation, boxplots
SciPy	1.10.1	Test statistique de Wilcoxon signé-rang
Kaggle	1.5.16	Gestion des datasets et soumission

Tableau 3. 1: Bibliothèques Python utilisées dans l'implémentation

L'intégration conjointe de scikit-learn et TensorFlow/Keras a permis d'assurer une cohérence dans nos processus de traitement des données tout en exploitant les capacités avancées de deep learning pour la mise en place de l'Autoencodeur. La bibliothèque SHAP, développée par Lundberg et Lee [32], constitue un outil d'interprétabilité reconnu, dont l'intégration dans notre pipeline répond directement à l'un des principaux défis de l'apprentissage non supervisé : l'explication des décisions du modèle

3.2.3 Reproductibilité et fixation du SEED :

Un aspect essentiel de notre démarche concerne la reproductibilité expérimentale. Afin de réduire la variabilité des résultats, une graine aléatoire fixe (SEED = 42) a été uniformément appliquée à l'ensemble des générateurs pseudo-aléatoires utilisés dans les différentes bibliothèques. Cette approche vise à garantir la stabilité et la cohérence des expériences dans un contexte académique.

Il est toutefois important de souligner que, malgré ces précautions, un léger caractère non déterministe peut subsister, notamment lors de l'entraînement des modèles neuronaux sur GPU. Ce phénomène est dû à la nature parallèle des calculs en virgule flottante, dont l'ordre d'exécution peut fluctuer, affectant ainsi la reproductibilité optimale. Néanmoins, l'impact de cette variabilité demeure marginal et ne remet pas en cause l'interprétation globale des résultats expérimentaux.

3.3 PRESENTATION ET ANALYSE DU JEU DE DONNEES CICMalDroid 2020 :

3.3.1 Origine et Processus de Collecte :

Dans tout système d'apprentissage automatique, le jeu de données joue un rôle fondamental, car la qualité et la représentativité des données influencent directement la pertinence des résultats obtenus. Dans le cadre de ce travail, nous nous appuyons sur le dataset **CICMalDroid 2020**, reconnu comme une référence dans le domaine de la cybersécurité mobile [12]. Cet ensemble de données a été élaboré par *le Canadian Institute for Cybersecurity (CIC)* de l'Université du Nouveau-Brunswick (UNB), selon une méthodologie rigoureuse fondée sur une analyse dynamique instrumentée, contrairement aux approches classiques basées sur l'analyse statique du code. il s'agit de l'un des jeux de données Android les plus complets et les plus largement utilisés dans la littérature scientifique, avec de nombreuses contributions académiques s'y référant depuis sa mise à disposition publique en 2020 [12].

Le processus de collecte repose sur l'exécution contrôlée d'applications Android (fichiers APK) au sein d'un environnement d'émulation instrumenté, permettant l'observation fine de leur comportement en conditions quasi réelles. Lors de cette phase, les interactions entre l'application et le système d'exploitation sont capturées sous forme d'appels système (*system calls* ou *syscalls* - *figure 3.1*), interceptés et enregistrés à l'aide d'outils spécialisés tels que strace.

Chaque syscall représente une requête d'une application vers le noyau Linux sous-jacent à Android, couvrant des opérations fondamentales telles que l'accès aux fichiers, la gestion de la mémoire ou les communications réseau. Afin de rendre ces données exploitables par les modèles d'apprentissage, la fréquence d'occurrence de chaque syscall est calculée sur une fenêtre d'observation donnée, constituant ainsi un vecteur de caractéristiques numériques représentatif du comportement de l'application.

Cette approche comportementale présente un avantage décisif sur les méthodes statiques. En effet, elle permet de capturer ce que l'application fait réellement au moment de l'exécution, et non simplement ce que son code source laisse supposer. Un malware sophistiqué peut dissimuler ses intentions dans son code (obfuscation, chiffrement du payload), mais ses interactions avec le système d'exploitation trahissent inévitablement son comportement réel [34].

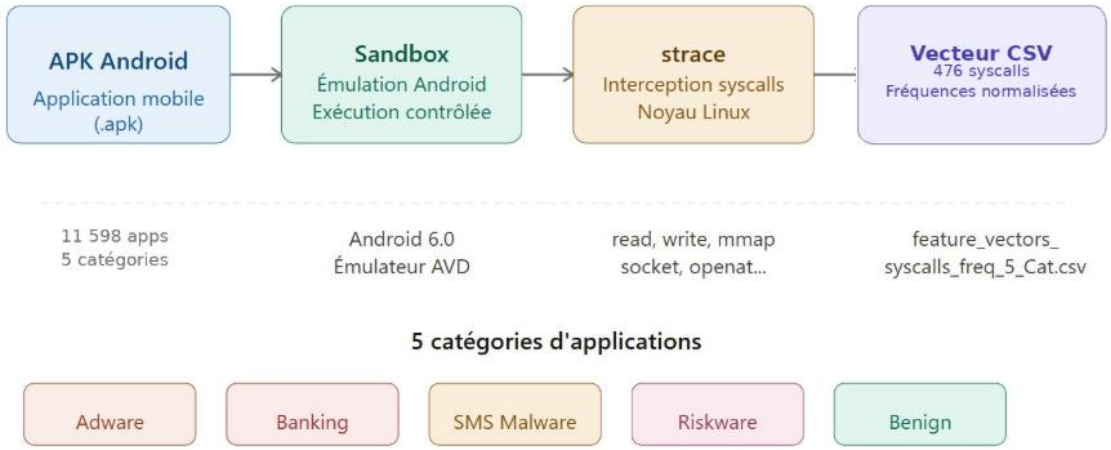


Figure 3. 1:Pipeline de collecte dynamique des syscalls (CICMalDroid 2020). (Produit par les auteurs, adapté de Rahali et al.) [12]

3.3.2 Structure et Caractéristiques du Dataset :

Le fichier utilisé dans notre implémentation est ***feature_vectors_syscalls_frequency_5_Cat.csv***, qui contient les vecteurs de fréquences des appels système (syscalls) pour l'ensemble des applications analysées. Ces applications sont réparties en cinq catégories distinctes, comprenant une classe bénigne et quatre classes de malwares. Après chargement, le dataset présente les dimensions suivantes : 11 598 échantillons × 477 colonnes (476 features syscalls + 1 colonne classe).

La distribution des classes au sein du dataset est présentée dans le tableau 3.2 ci-dessous :

Classe	Type	Échantillons	Proportion (%)	Encodage binaire
Benign	Sain	1250	10.8%	0 (normal)
Adware	Malware	2100	18.1%	1 (anomalie)
Banking	Malware	3904	33.7%	1 (anomalie)
SMS	Malware	2546	22.0%	1 (anomalie)

Riskware	Malware	1795	15.5%	1 (anomalie)
Total	—	11598	100%	—

Tableau 3. 2:Distribution des classes dans CICMalDroid 2020

Plusieurs observations importantes se dégagent de cette distribution. Tout d'abord, le dataset présente un fort déséquilibre entre les classes : les applications malveillantes représentent environ 89.2 % des échantillons, contre 10.8 % d'applications bénignes. Ce déséquilibre est en réalité représentatif des conditions réelles que l'on retrouve dans les stores d'applications alternatifs [35], ce qui renforce la validité écologique de nos expériences. Par ailleurs, la classe Banque constitue la catégorie la plus représentée avec **33,7 %** des échantillons. Cette domination s'explique par la forte activité des malwares bancaires dans l'écosystème Android, ces derniers ciblant spécifiquement les applications financières afin d'intercepter des données sensibles telles que les identifiants ou les codes d'authentification. À l'inverse, la catégorie Riskware regroupe des applications dont le comportement est potentiellement risqué sans révéler malveillant, ce qui les rend plus difficiles à distinguer des applications bénignes lors de l'analyse comportementale.

Chaque échantillon du jeu de données est décrit par un ensemble de 476 caractéristiques numériques, représentant les fréquences normalisées des appels système observés lors de l'exécution. Ces caractéristiques sont dérivées d'environ 470 types distincts de syscalls, comprenant s'ajoutent des variables complémentaires liées à l'identification ou au prétraitement des données. Les fréquences sont exploitées comme le rapport entre le nombre d'occurrences d'un appel système donné et le nombre total de syscalls générés par l'application, ce qui permet d'obtenir une représentation indépendante de la durée d'exécution et comparable entre les différentes applications.

La Figure 3.2 illustre la distribution des cinq classes du dataset CICMalDroid 2020. La classe **Banking** est la plus représentée avec 3 904 échantillons (33,7 %), suivie des classes **SMS** (22,0 %), **Adware** (18,1 %) et **Riskware** (15,5 %). La classe **Benign**, représentant les applications légitimes, est la moins représentée avec 1 253 échantillons (10,8 %). Cette répartition révèle un déséquilibre modéré entre les classes, susceptible d'influencer le processus d'apprentissage et

les performances des modèles de classification. Néanmoins, ce déséquilibre demeure relativement limité et permet une évaluation fiable des approches de détection étudiées.

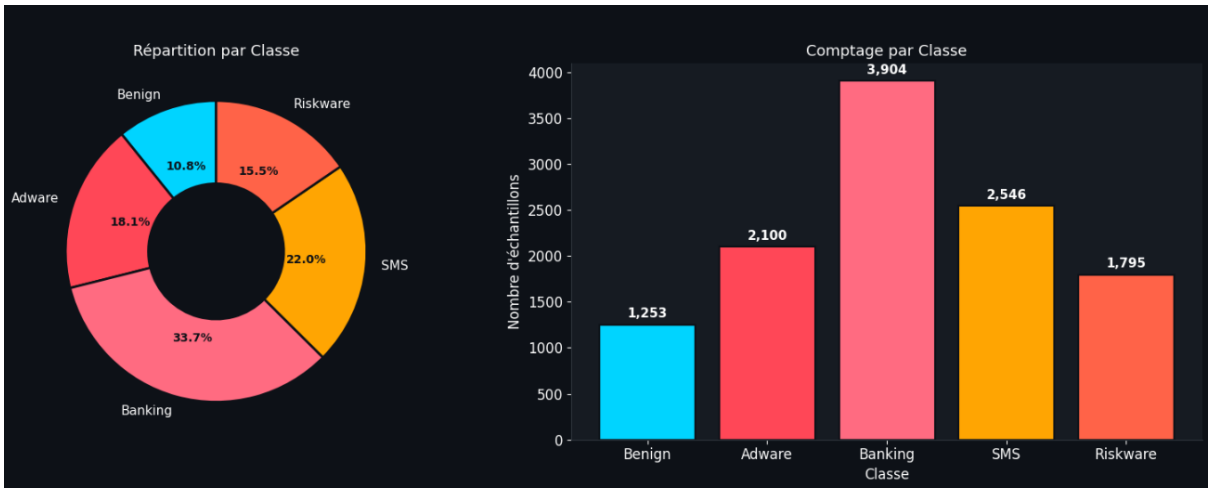


Figure 3. 2:Distribution des cinq classes du dataset CICMalDroid 2020.

3.3.3 Analyse Exploratoire des Données (EDA) :

Avant toute phase de modélisation, une analyse exploratoire rigoureuse est indispensable pour comprendre la structure des données, identifier ses caractéristiques principales et détecter d'éventuelles anomalies ou incohérences pouvant influencer les performances des modèles d'apprentissage automatique. Cette étape constitue une phase essentielle dans tout pipeline de Machine Learning, car elle permet d'orienter les choix de prétraitement, de réduction de dimensionnalité et de sélection des modèles. L'analyse exploratoire du dataset CICMalDroid 2020 a porté sur plusieurs aspects, notamment la distribution des classes, les propriétés statistiques des features, les corrélations entre variables ainsi que la visualisation globale des données.

3.3.3.1 Analyse statistique descriptive :

Dans cette partie, une étude statistique des variables du dataset a été réalisée afin d'identifier les caractéristiques principales des données. Les résultats montrent que la majorité des features présentent des valeurs faibles et fortement concentrées autour de zéro, ce qui traduit une structure de données creuse (*sparse data*). Certains appels système tels que *read ()*, *write ()* et *connect ()* apparaissent plus fréquemment en raison de leur rôle essentiel dans les opérations d'entrée/sortie et les communications réseau des applications Android.

3.3.3.2 Distribution des classes et déséquilibre :

L'analyse de la distribution des classes met en évidence le fort déséquilibre mentionné précédemment. Les différentes catégories de malwares représentent la grande majorité des échantillons du dataset (89,2 %), tandis que la classe Benign reste faiblement représentée (10,8%). Cette caractéristique reflète les conditions réelles rencontrées dans certains environnements Android non contrôlés et justifie le recours aux approches basées sur la détection d'anomalies plutôt qu'à la classification supervisée classique.

3.3.3.3 Analyse des corrélations :

La heatmap de corrélation entre les features syscalls (figure 3.3) révèle l'existence de groupes de syscalls fortement corrélés entre eux. Cette redondance informationnelle justifie l'application d'une réduction de dimensionnalité par PCA, qui permettra de compresser l'information tout en préservant l'essentiel de la variance. En particulier, les syscalls liés aux opérations de lecture/écriture (read, write, pread64, pwrite64) forment un cluster très corrélé, de même que les syscalls réseau (socket, connect, recvfrom, sendto).

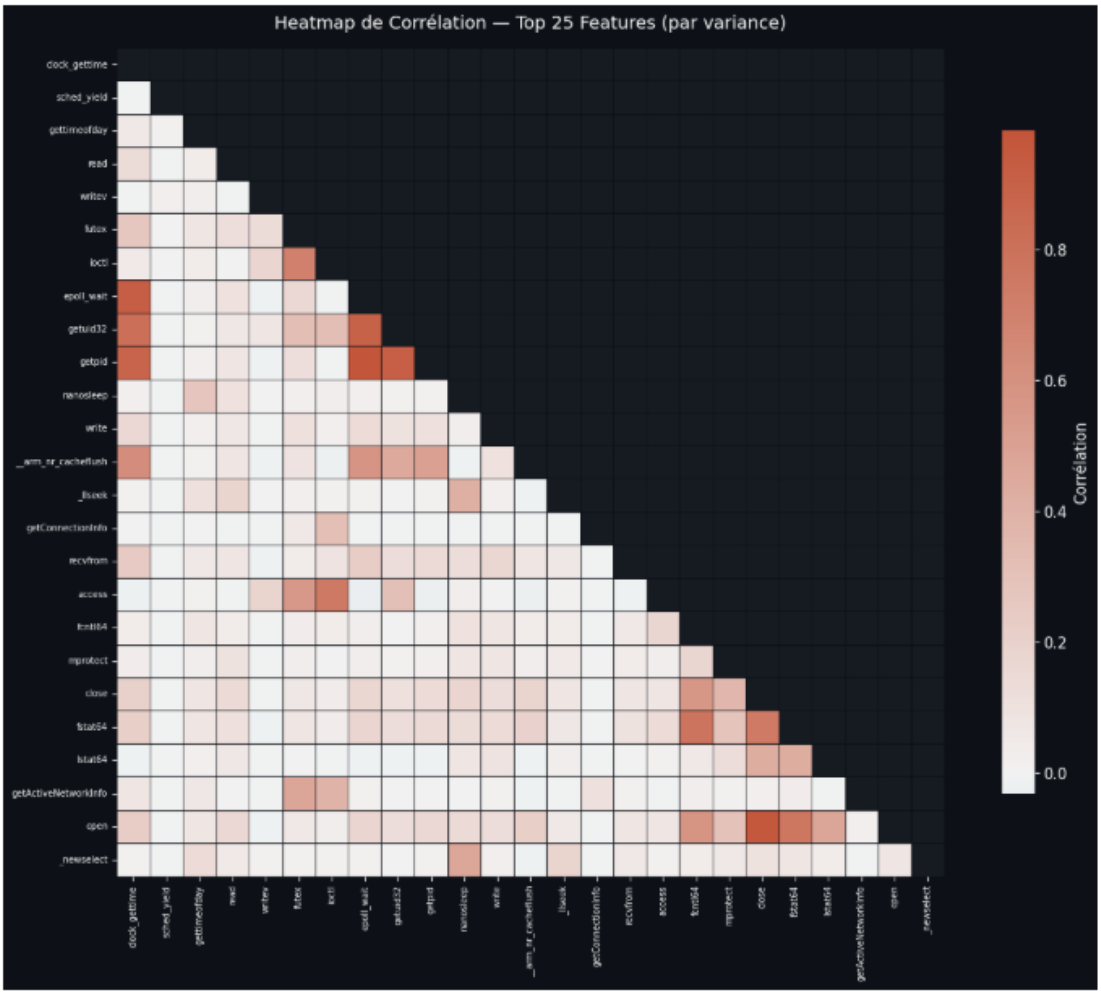


Figure 3. 3:Heatmap de corrélation des 25 features.

3.3.3.4 Visualisation des données :

Des techniques de visualisation ont été appliquées afin d'obtenir une représentation globale du dataset. La méthode t-SNE a notamment été utilisée pour projeter les données dans un espace bidimensionnel après réduction de dimensionnalité. Les résultats montrent une séparation partielle entre les applications bénignes et certaines catégories de malwares, bien qu'un chevauchement demeure pour des classes comme *Riskware*, soulignant ainsi la complexité du problème de détection étudié.

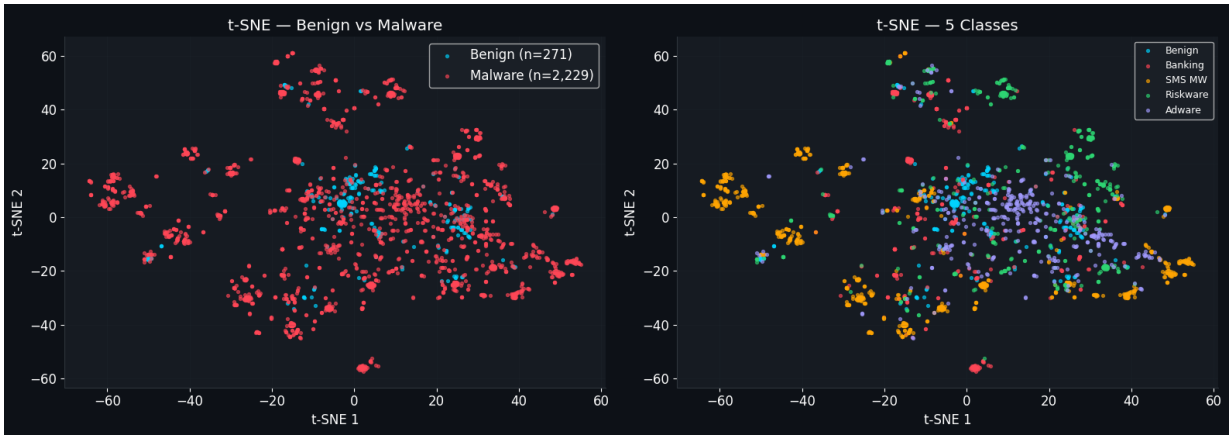


Figure 3. 4: t-SNE 2D structure Non Linéaire du dataset

Figure 3.4 présente une projection en 2D d'un jeu de données à l'aide de t-SNE, une technique de réduction de dimensionnalité non linéaire. Elle est divisée en deux graphiques qui analysent la distribution des échantillons selon différents niveaux de précision.

- Graphique de gauche : Perspective de classification binaire (Benign vs. Malware).
- Graphique de droite : Perspective multiclasse répartie en 5 catégories spécifiques (Benign, Banking, SMS MW, Riskware, Adware).

3.4 JUSTIFICATION DU CHOIX DES MODELES :

La sélection des algorithmes de détection d'anomalies retenus dans le cadre de cette étude s'est effectuée après une analyse approfondie prenant en considération plusieurs critères : la nature non supervisée du problème, les caractéristiques intrinsèques du jeu de données CICMalDroid 2020, les contraintes opérationnelles, ainsi que l'état de l'art en matière de détection de malwares Android. Trois modèles ont ainsi été choisis, chacun représentant un paradigme algorithmique différent et complémentaire.

3.4.1 Isolation Forest (Détection par Isolation Stochastique) :

Dans ce travail, le choix de l'algorithme Isolation Forest repose principalement sur son adéquation avec les caractéristiques du dataset CICMalDroid 2020 et avec la nature du problème de détection d'anomalies étudié. En effet, le dataset comporte 470 caractéristiques issues des fréquences d'appels système Android, ce qui correspond à un espace de grande dimension dans lequel les comportements malveillants peuvent apparaître comme des observations atypiques.

Le choix de l'algorithme Isolation Forest dans ce travail est justifié par plusieurs avantages:

- Sa faible complexité computationnelle permet un entraînement rapide, même sur des ensembles de données volumineux.
- Il ne suppose aucune distribution statistique spécifique pour les données, ce qui est essentiel puisque les fréquences des appels système ne suivent pas une distribution gaussienne classique.
- Il fournit un score d'anomalie continu pour chaque application, facilitant ainsi l'ajustement du seuil de détection.
- Il est capable de détecter efficacement les comportements inhabituels associés aux malwares Android, tels que certains appels système sensibles ou anormaux.
- Il présente une robustesse appréciable face à des données de haute dimension et à des comportements malveillants inconnus (zero-day).[36]

3.4.2 One-Class SVM (Modélisation Frontière à Noyau RBF) :

Le choix du modèle One-Class SVM dans le cadre de ce travail est motivé par sa capacité à définir une frontière de décision autour des comportements considérés comme normaux, en se basant uniquement sur des applications bénignes. Cette propriété en fait une solution particulièrement adaptée à la détection d'anomalies dans des contextes de cybersécurité, notamment pour identifier des malwares inconnus ou de type zero-day, qui ne sont pas présents dans l'ensemble d'entraînement.[37]

Plusieurs éléments justifient l'intégration de ce modèle dans notre protocole expérimental :

- Il constitue l'un des modèles de référence les plus utilisés en apprentissage one-class, permettant une comparaison fiable avec les autres approches.
- L'utilisation du noyau RBF (Radial Basis Function) permet de modéliser des frontières de décision non linéaires adaptées à la complexité des vecteurs de syscalls Android.
- Le paramètre ν permet de contrôler directement le taux d'anomalies attendu et d'ajuster la sensibilité du modèle selon les besoins de détection.
- Le modèle est particulièrement efficace pour distinguer les comportements normaux des comportements atypiques dans des espaces de grande dimension.
- Il s'adapte bien aux caractéristiques du dataset CICMalDroid 2020 composé de 470 variables liées aux appels système Android.

Cependant, le One-Class SVM présente également certaines limitations :

- Sa complexité computationnelle élevée augmente fortement avec la taille des données.
- Dans l'Approche 2 utilisant les données rééquilibrées par SMOTETomek, le temps d'entraînement atteint 72,2 secondes, ce qui représente le coût computationnel le plus important parmi les modèles étudiés.
- Le modèle est sensible à l'échelle des données, ce qui nécessite une normalisation rigoureuse avant l'entraînement.

3.4.3 Autoencodeur (Représentation Latente et Reconstruction) :

Le choix de l'Autoencodeur dans ce travail est motivé par sa capacité à apprendre automatiquement une représentation compacte des comportements normaux à partir des données bénignes. Grâce à son architecture encodeur-décodeur, le modèle apprend à reconstruire les applications légitimes, tandis que les applications malveillantes génèrent généralement une erreur de reconstruction plus élevée, utilisée comme indicateur d'anomalie.[38]

Plusieurs éléments justifient l'utilisation de cette approche :

- Sa capacité à apprendre des représentations latentes non linéaires adaptées aux données de grande dimension contenant 470 caractéristiques.
- Son pouvoir expressif élevé comparé aux méthodes classiques linéaires.
- La possibilité d'intégrer des mécanismes de régularisation comme le Dropout et la régularisation L2 afin de limiter le surapprentissage.
- L'utilisation de l'erreur de reconstruction comme score d'anomalie, offrant une interprétation plus intuitive des décisions du modèle.
- Sa capacité à extraire automatiquement les caractéristiques importantes sans nécessiter d'ingénierie manuelle complexe.

L'architecture retenue dans cette étude est une architecture symétrique profonde de type :

470→256→128→64→32→64→128→256→470

Cette structure a été choisie empiriquement afin d'obtenir un compromis entre compression de l'information et qualité de reconstruction. Dans l'Approche 2, la taille du bottleneck est réduite à 16 neurones afin d'imposer une compression plus forte et d'améliorer la séparation

entre comportements normaux et malveillants. De plus, une pondération asymétrique des classes a été appliquée durant l'entraînement afin de spécialiser le modèle dans la reconstruction des applications bénignes.

3.5 PRETRAITEMENT DES DONNEES :

Le pipeline de prétraitement adopté dans cette étude est composé de quatre étapes séquentielles, exécutées dans un ordre rigoureusement défini afin de garantir l'intégrité expérimentale et d'éviter tout phénomène de fuite d'information (data leakage) entre les ensembles d'entraînement et de test [39]. Cette organisation méthodologique permet de préserver la capacité de généralisation des modèles en s'assurant que les informations issues des données de test ne soient jamais exploitées durant la phase d'apprentissage. De plus, cette stratégie contribue à produire des évaluations plus fiables et représentatives des performances réelles du système de détection.

3.5.1 Encodage de la variable cible :

Dans le cadre de cette étude, nous avons exploité le jeu de données CICMalDroid 2020, dont la variable cible « Class » initialement catégorisée en cinq classes : Benign, Adware, Banking, SMS et Riskware. Afin de nous concentrer sur la détection d'anomalies, cette représentation a été simplifiée en une classification binaire.

Concrètement, toutes les instances de la classe « Benign » ont été codées par y=0, correspondant à un comportement normal, tandis que l'ensemble des catégories malveillantes a été regroupé en une seule classe codée par y=1, représentant un comportement anormal :

$$\begin{cases} y = 0 \; si \; \langle class \rangle = benign \\ y = 1 \; si \; \langle class \rangle \in \{Adware, Banking, SMS, Riskware\} \end{cases}$$

Cette approche permet de transformer le problème en une tâche de détection d'anomalies, visant à différencier les comportements légitimes des comportements malveillants. Elle est particulièrement adaptée aux méthodes d'apprentissage en une classe (« one-class learning ») et aux enjeux de la cybersécurité, où les attaques sont rares, en constante évolution ou parfois inconnues (attaques de type zero-day). Par ailleurs, le codage binaire facilite l'évaluation des performances à l'aide de métriques telles que la précision, le rappel et le score F1.

3.5.2 Séparation Train / Validation / Test :

La stratégie de découpe des données diffère selon l'approche adoptée.

Pour l'approche 1(*one-class*) seules les 1 253 applications bénignes sont utilisées durant l'apprentissage, avec une répartition de 70 % pour l'entraînement (877 échantillons), 15 % pour la validation (188 échantillons) et 15 % pour le test (188 échantillons). L'ensemble de test final est ensuite constitué de ces 188 applications bénignes ainsi que de l'ensemble des 10 345 applications malveillantes, soit 10 533 échantillons au total, afin de reproduire un scénario réaliste de détection d'anomalies.

Dans l'approche 2 (semi-supervisée) une découpe stratifiée est réalisée avant toute opération de rééquilibrage : 70 % des données sont affectées à l'entraînement (8 118 échantillons), 15 % à la validation (1 740 échantillons) et 15 % au test (1 740 échantillons). Cette procédure est essentielle pour éviter toute fuite d'information liée à l'utilisation de SMOTE, en empêchant que des échantillons synthétiques apparaissent dans les ensembles de validation ou de test [40].

3.5.3 Normalisation par StandardScaler :

Les 470 caractéristiques du jeu de données ont été normalisées à l'aide de la standardisation Z-score, une méthode qui transforme chaque variable afin qu'elle possède une moyenne nulle et un écart-type unitaire. Les paramètres de normalisation, à savoir la moyenne et l'écart-type, ont été calculés uniquement à partir de l'ensemble d'entraînement afin d'éviter toute fuite d'information vers les ensembles de validation et de test. Cette étape est particulièrement importante pour les modèles sensibles à l'échelle des données, notamment le One-Class SVM utilisant un noyau RBF. [41]

Par ailleurs, une normalisation complémentaire à l'aide de MaxAbsScaler a été appliquée spécifiquement au One-Class SVM. Cette technique permet de ramener les valeurs des caractéristiques dans l'intervalle [-1, 1] tout en conservant la structure des données et les valeurs nulles. Cette approche est recommandée dans scikit-learn afin d'améliorer la stabilité numérique et les performances du modèle.

3.5.4 Rééquilibrage par SMOTE (approche 2 uniquement) :

Dans le cadre de l'approche semi-supervisée, le déséquilibre important entre les classes a été adressé à l'aide de la méthode hybride SMOTETomek [42], appliquée exclusivement à l'ensemble d'entraînement afin d'éviter toute fuite d'informations. Cette technique combine l'algorithme SMOTE (Synthetic Minority Over-sampling Technique) et la méthode de nettoyage Tomek Links.

SMOTE est une technique de suréchantillonnage qui permet d'accroître artificiellement le nombre d'exemples de la classe minoritaire sans simplement dupliquer les données existantes. Son principe consiste à sélectionner, pour chaque échantillon minoritaire, ses k plus proches voisins dans l'espace des caractéristiques, puis à générer de nouveaux exemples synthétiques par interpolation linéaire entre ces points. Ainsi, les nouvelles observations créées restent proches des données réelles tout en introduisant une variabilité supplémentaire dans l'espace de représentation. Cette approche améliore la capacité de généralisation du modèle et réduit le risque de surapprentissage associé à la duplication simple des échantillons minoritaires.

Cependant, SMOTE peut également entraîner la génération d'observations synthétiques dans des zones où les classes se chevauchent. Afin de limiter cet effet, la méthode Tomek Links est appliquée après le suréchantillonnage pour identifier et supprimer les paires d'exemples ambigus situées à proximité de la frontière de décision. Cette étape contribue au nettoyage des données et à une meilleure séparation entre les classes.

Suite à l'application de SMOTETomek, le nombre d'exemples bénins est passé de 877 à 7 240, permettant de constituer un ensemble d'entraînement équilibré entre classe bénigne et malveillante (50 % / 50 %), totalisant ainsi 14 480 échantillons, dont 6 362 sont synthétiques.

3.5.5 Réduction de dimensionnalité par PCA :

Bien que la réduction de dimensionnalité par l'Analyse en Composantes Principales (ACP ou PCA) n'ait pas été intégrée directement dans le processus d'entraînement des modèles finaux ces derniers utilisant l'ensemble complet des 470 caractéristiques initiales une étude exploratoire utilisant un ACP à 50 composantes principales a été menée à des fins d'analyse et de visualisation.

L'ACP est une technique statistique permettant de réduire la complexité d'un ensemble de variables potentiellement corrélées en le projetant dans un espace de dimensions inférieures constitué de composantes principales orthogonales. Ces composantes sont déterminées de manière à maximiser la variance expliquée, la première composante capturant l'essentiel de l'information, suivie par les suivantes selon un ordre d'importance décroissant. Cette méthode permet de condenser l'information essentielle contenue dans des données à haute dimension tout en atténuant le bruit et la redondance.

Dans le cadre de cette étude, l'application d'un ACP à 50 composantes a permis de préserver 48,9 % de la variance totale des données, offrant ainsi une représentation synthétique du jeu de

données tout en conservant une part substantielle de sa structure statistique. Les projections obtenues ont été principalement utilisées pour réaliser des visualisations en deux dimensions (2D), facilitant l'analyse visuelle de la séparation entre comportements bénins et malveillants.

Cependant, malgré son utilité pour l'exploration et la visualisation, l'intégration de l'ACP lors de l'entraînement final des modèles n'a pas été retenue. En effet, certaines caractéristiques discriminantes présentes dans l'ensemble original pouvaient se perdre lors de la projection dans un espace réduit, risquant ainsi de dégrader la performance des modèles de détection d'anomalies. Pour cette raison, toutes les expérimentations finales ont été effectuées en utilisant l'ensemble complet des 470 caractéristiques initiales.

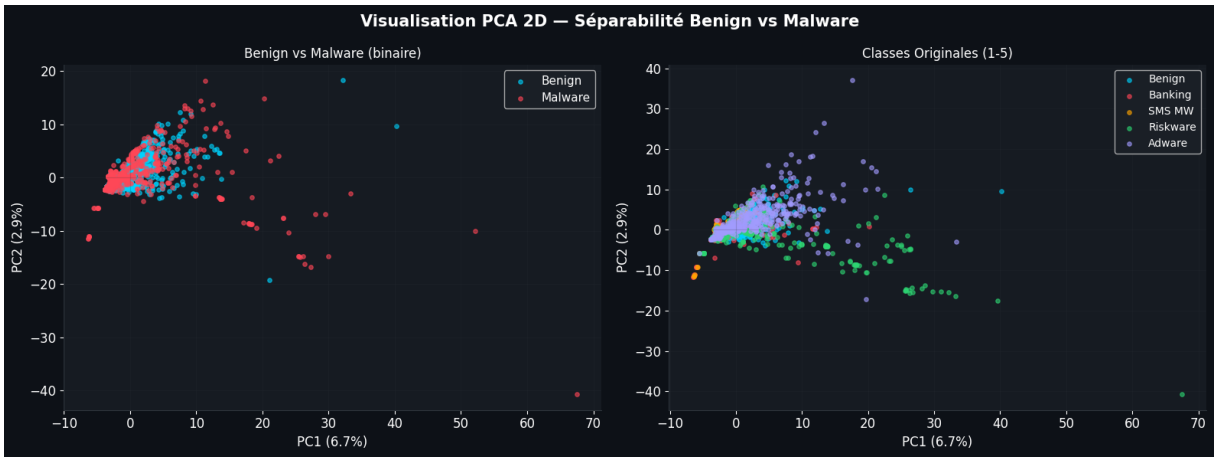


Figure 3. 5:Visualisation PCA 2D. Séparabilité Benign vs Malware.

Figure 3.5 montre une projection linéaire du même jeu de données à l'aide de la PCA (ou ACP). Contrairement au t-SNE (qui préserve les distances locales et non linéaires), la PCA cherche à projeter les données selon les axes de variance maximale.

- Axe X (PC1) : Première composante principale, qui n'explique que 6,7% de la variance totale.
- Axe Y (PC2) : Deuxième composante principale, qui n'explique que 2,9% de la variance totale.
- Variance cumulée (PC1 + PC2) : Ensemble, ces deux axes ne capturent que 9,6% de la variance des données. Cela indique d'emblée que le jeu de données est très complexe, multidimensionnel, et qu'une projection linéaire en 2D perd une immense quantité d'informations (plus de 90%).

3.5.6 Métriques d'Évaluation :

Les performances des modèles proposés sont évaluées à l'aide de plusieurs métriques complémentaires, adaptées au contexte spécifique de la détection d'anomalies et au déséquilibre important du jeu de données. Chaque métrique permet d'analyser un aspect particulier de la performance du système :

- **Accuracy (Exactitude)** : elle indique la proportion globale de prédictions correctes. Bien qu'elle soit moins pertinente dans un contexte déséquilibré, elle constitue néanmoins un indicateur de référence général.
- **Précision** : elle mesure la proportion de détections d'applications malveillantes qui sont effectivement des malwares. Cette métrique permet d'évaluer la fiabilité des alertes émises par le système.
- **Recall (Rappel ou Sensibilité)** : il quantifie la capacité du modèle à détecter correctement les malwares réels. En cybersécurité, cette métrique est cruciale, car les faux négatifs correspondent à des menaces non identifiées.
- **F1-Score** : il s'agit de la moyenne harmonique entre la précision et le rappel, permettant de trouver un équilibre optimal entre la détection des attaques et la réduction des fausses alertes.
- **AUC-ROC (Area Under the Curve - Receiver Operating Characteristic)** : cette métrique évalue la capacité globale du modèle à différencier les applications légitimes des applications malveillantes, indépendamment du seuil de décision choisi.

Pour optimiser les performances des modèles, un ajustement du seuil de décision est effectué à partir de l'analyse de la courbe Precision-Recall. Cette approche vise à maximiser le score F1 tout en maintenant un niveau élevé de rappel, ce qui est essentiel dans un contexte de détection de malwares caractérisé par un déséquilibre important des classes.

3.6 PROTOCOLE EXPERIMENTAL :

Notre protocole expérimental est structuré autour de deux approches complémentaires, conçues pour répondre à des questions scientifiques distinctes tout en s'appuyant sur les mêmes modèles et le même dataset.

3.6.1 Approche 1 (AP1) (Détection One-Class) :

La première approche expérimentale adoptée dans ce travail repose sur une stratégie de détection *One-Class*, elle simule une situation de déploiement réel dans laquelle le système de

détection est entraîné uniquement sur des applications connues comme saines, sans aucune connaissance préalable des malwares. Ce scénario, couramment désigné sous le terme de détection d'anomalies en mode zéro-coup (zero-shot anomaly detection) dans la littérature [10], revêt une importance particulière dans le domaine de la cybersécurité mobile, où de nouvelles familles de logiciels malveillants apparaissent fréquemment.

Dans cette configuration, après la séparation du dataset en ensembles d'entraînement et de test, seuls les échantillons appartenant à la classe *Benign* ont été utilisés pour l'apprentissage des modèles Isolation Forest, One-Class SVM et Autoencodeur. Les données de test contiennent quant à elles à la fois des applications bénignes et des applications malveillantes issues des différentes catégories du dataset CICMalDroid 2020.

3.6.1.1 Résultats de l'Isolation Forest (AP1) :

L'Isolation Forest entraîné sur les bénins uniquement (n_estimators=1000, max_features=0.7, bootstrap=True) a été ajusté avec le seuil optimal (seuil = 0.320635) obtenu par recherche sur la courbe Précision-Rappel.

Métrique	**Valeur brute (contam=0.05)**	**Valeur optimisée (seuil adaptatif)**
Accuracy	0.1252	0.9835
Precision	0.9922	0.9835
Recall	0.1102	1.0000
F1-Score	0.1984	0.9917
AUC-ROC	0.6000	0.6000
Temps entraînement	—	3,2 s

Tableau 3. 3: Résultats Isolation Forest (Approche 1 (Benign only))

L'optimisation du seuil de décision transforme radicalement les performances : le Recall passe de 11,02 % à 100,00 % tout en maintenant une Précision de 98,35 %. Ce résultat

remarquable (Tableau 3.3) s'explique par la capacité de l'Isolation Forest à attribuer des scores d'anomalie très élevés aux malwares, dont les patterns comportementaux (séquences de syscalls) sont structurellement différents de ceux des bénins. La valeur d'AUC-ROC (0.6000) reste néanmoins modérée, indiquant que la séparabilité globale des scores n'est pas parfaite, mais que le seuil optimal identifié permet d'exploiter efficacement cette séparabilité partielle.

3.6.1.2 Résultats du One-Class SVM (AP1) :

Le One-Class SVM à noyau RBF (nu=0.02, gamma='scale') a été entraîné en 0,0 seconde. Le seuil optimal (−1.010018)

Métrique	Valeur brute (nu=0.02)	Valeur optimisée (seuil adaptatif)
Accuracy	0.3395	0.9822
Precision	0.9956	0.9822
Recall	0.3290	1.0000
F1-Score	0.4945	0.9910
AUC-ROC	0.7687	0.7687
Temps entraînement	—	< 0,1 s

Tableau 3. 4:Résultats One-Class SVM (Approche 1 (Benign only))

Le One-Class SVM présente l'AUC-ROC la plus élevée de l'Approche 1 (0.7687), ce qui indique une meilleure séparabilité intrinsèque des scores par rapport à l'Isolation Forest. Après optimisation du seuil, le modèle atteint également un Recall de 100 % avec une Précision de 98,22 % (Tableau 3.4). La rapidité d'entraînement (< 0,1 s) constitue un avantage majeur pour un déploiement en conditions réelles.

3.6.1.3 Résultats de l'Autoencoder (AP1) :

L'Autoencoder profond (architecture 470→256→128→64→32→64→128→256→470, 329 078 paramètres) a été entraîné pendant 100 époques (avec arrêt précoce) sur les 877 bénins d'entraînement. Le meilleur val_loss a été atteinte avant convergence complète, révélant un

phénomène de sur-apprentissage (gap train/val de 732,9 %). Le seuil de détection optimal (0.002962) est déterminé à partir de la distribution des erreurs MSE sur les bénins d'entraînement.

Métrique	Valeur
Accuracy	0.9833
Precision	0.9835
Recall	0.9998
F1-Score	0.9916
AUC-ROC	0.5807 [IC 95% : 0.5407–0.6195]
Seuil MSE optimal	0.002962
Temps entraînement	18,5 s

Tableau 3. 5:Résultats Autoencoder (Approche 1 (Benign only))

L'Autoencoder obtient une Précision de 98,35 % et un Recall de 99,98 %, confirmant sa capacité à distinguer les bénins des malwares via l'erreur de reconstruction (Tableau 3.5). L'AUC-ROC (0.5807) est plus faible que celle du One-Class SVM, car la distribution des erreurs MSE se chevauche partiellement entre bénins et malwares pour des seuils intermédiaires. La forte erreur de reconstruction observée pour les malwares s'explique par le fait que l'Autoencodeur a appris une représentation compacte spécifique aux comportements bénins.

3.6.2 Approche 2 (Validation croisée semi-supervisée (SMOTETomek)) :

Contrairement à l'approche AP1 basée sur un apprentissage exclusivement centré sur les applications bénignes, l'Approche 2 expose les modèles aux deux classes pendant la phase d'entraînement, à savoir les applications bénignes et les applications malveillantes. Cette stratégie vise à fournir aux modèles une représentation plus équilibrée des comportements

présents dans l'ensemble de données, afin d'améliorer leur capacité de discrimination entre comportements normaux et comportements anormaux.

Dans notre implémentation, les données originales ont été séparées selon une stratégie stratifiée avec une répartition de 70 % pour l'entraînement, 15 % pour la validation et 15 % pour le test. L'ensemble d'entraînement obtenu contient initialement 8 118 échantillons, répartis entre 877 applications bénignes et 7 241 applications malveillantes.

En raison du fort déséquilibre entre les classes, une technique de rééquilibrage hybride basée sur SMOTETomek a été appliquée uniquement sur l'ensemble d'entraînement. Cette méthode combine l'algorithme SMOTE (*Synthetic Minority Over-sampling Technique*), permettant de générer artificiellement de nouveaux échantillons de la classe minoritaire, et la méthode Tomek Links, utilisée pour supprimer les observations ambiguës situées à la frontière entre les classes.

Le résultat est un ensemble d'entraînement de 14 480 échantillons équilibrés (7 240 bénins, 7 240 malwares). Les ensembles de validation et de test (1 740 échantillons chacun : 188 bénins + 1 552 malwares) ne contiennent aucun échantillon synthétique, garantissant une évaluation sur données réelles uniquement. Cette stratégie permet d'assurer une évaluation plus réaliste et plus fiable des performances des modèles sur des données réelles non artificielles.

3.6.2.1 Résultats de l'Isolation Forest (AP2) :

L'Isolation Forest de l'Approche 2 (Tableau 3.6 - contamination=0.5, max_samples=1024) a été entraîné sur les données rééquilibrées en 25,0 secondes. Le seuil optimal conduit à un Recall de 96,52% et une Précision de 90,57 %.

Métrique	Valeur brute	Valeur optimisée
Accuracy	0.5540	0.8793
Precision	0.9101	0.9057
Recall	0.5548	0.9652
F1-Score	0.6894	0.9345

AUC-ROC	0.5961	0.5961
Temps entraînement	—	25,0 s

Tableau 3. 6:Résultats Isolation Forest (Approche 2 (SMOTETomek))

3.6.2.2 Résultats du One-Class SVM (AP2) :

Le One-Class SVM de l'Approche 2 (Tableau 3.7 - nu=0.50, gamma='scale') a été entraîné en 72,2 secondes, ce qui représente le coût computationnel le plus élevé de l'étude (complexité quadratique en $O(n^2)$ avec n=14 480).

Métrique	Valeur brute	Valeur optimisée
Accuracy	0.6425	0.7943
Precision	0.9227	0.9129
Recall	0.6540	0.8505
F1-Score	0.7655	0.8806
AUC-ROC	0.6726	0.6726
Temps entraînement	—	72,2 s

Tableau 3. 7:Résultats One-Class SVM (Approche 2 (SMOTETomek))

3.6.2.3 Résultats de l'Autoencodeur (AP2) :

L'Autoencodeur de l'Approche 2 (architecture identique mais bottleneck réduit à 16 neurones, poids d'entraînement : bénins ×1.0, malwares ×0.15) a été entraîné pendant 150 époques en 81,4 secondes (Tableau 3.8). Cette pondération asymétrique force le modèle à se spécialiser dans la reconstruction des bénins, même en présence des malwares dans l'ensemble d'entraînement

Métrique	Valeur
Accuracy	0.8592

Precision	0.9036
Recall	0.9427
F1-Score	0.9227
AUC-ROC	0.6062
Temps entraînement	81,4 s

Tableau 3. 8: Résultats Autoencoder (Approche 2 (SMOTETomek))

3.7 RESULTATS ET DISCUSSION :

3.7.1 Tableau comparatif global :

Le tableau 3.9 présente une synthèse comparative des six configurations évaluées selon les cinq métriques retenues ainsi que les temps d'entraînement.

Modèle	Paradigme	Accuracy	Precision	Recall	F1-Score	AUC-ROC	Temps (s)
IF AP1	One-Class	0.9835	0.9835	1.0000	0.9917	0.6000	3,2
OC-SVM AP1	One-Class	0.9822	0.9822	1.0000	0.9910	0.7687	< 0,1
AE AP1	One-Class	0.9833	0.9835	0.9998	0.9916	0.5807	18,5
IF AP2	Semi-sup	0.8793	0.9057	0.9652	0.9345	0.5961	25,0
OC-SVM AP2	Semi-sup	0.7943	0.9129	0.8505	0.8806	0.6726	72,2
AE AP2	Semi-sup	0.8592	0.9036	0.9427	0.9227	0.6062	81,4

Tableau 3. 9: Synthèse comparative des six modèles (résultats avec seuil optimal)

Figure 3.6-Carte de chaleur (heatmap) représentant les métriques de performance des six configurations expérimentales étudiées. Les valeurs sont codées par couleur selon une échelle

allant du rouge (faible performance, proche de 0) au vert (performance élevée, proche de 1), permettant une comparaison visuelle rapide des modèles et des approches évaluées.

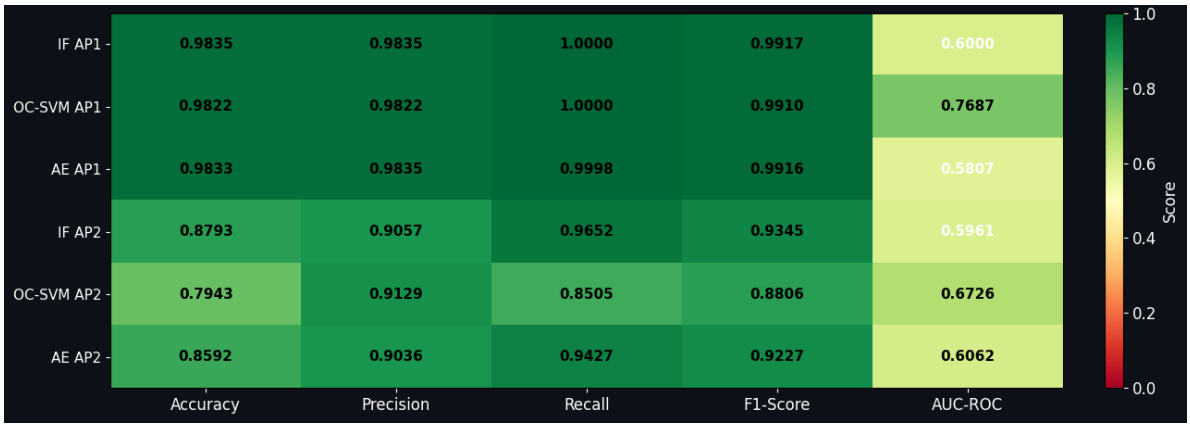


Figure 3. 6: Carte de chaleur (heatmap).

Figure 3.7-Diagramme radar (*radar chart*) illustrant la comparaison des cinq métriques de performance pour les six configurations expérimentales étudiées. Les résultats montrent que les modèles de l'Approche 1 dominent globalement selon les métriques de *Recall* et de *F1-Score*, traduisant une meilleure capacité de détection des anomalies. Par ailleurs, la configuration OC-SVM AP1 présente les meilleures performances selon la métrique AUC-ROC, indiquant une excellente capacité de discrimination entre applications bénignes et malveillantes.

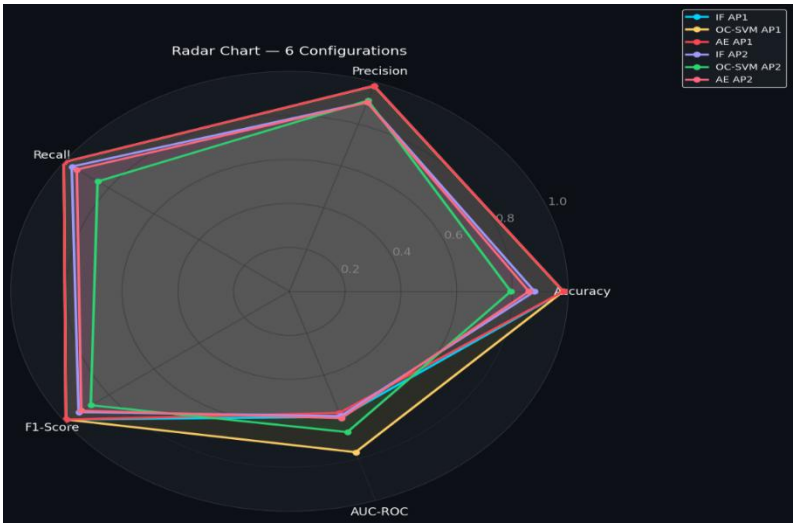


Figure 3. 7: Diagramme radar (radar chart).

Figure 3.8. represente une comparaison des temps d'entraînement des six modèles étudiés, exprimés en secondes. Les résultats montrent que le modèle OC-SVM AP1 présente le temps d'exécution le plus faible (moins de 0,1 seconde), ce qui traduit sa grande rapidité d'apprentissage. À l'inverse, l'Autoencodeur AP2 est le modèle le plus coûteux en calcul avec

un temps d'entraînement de 81,4 secondes, en raison de la complexité de son architecture neuronale et du processus d'optimisation itératif. Source : généré à partir du code expérimental.

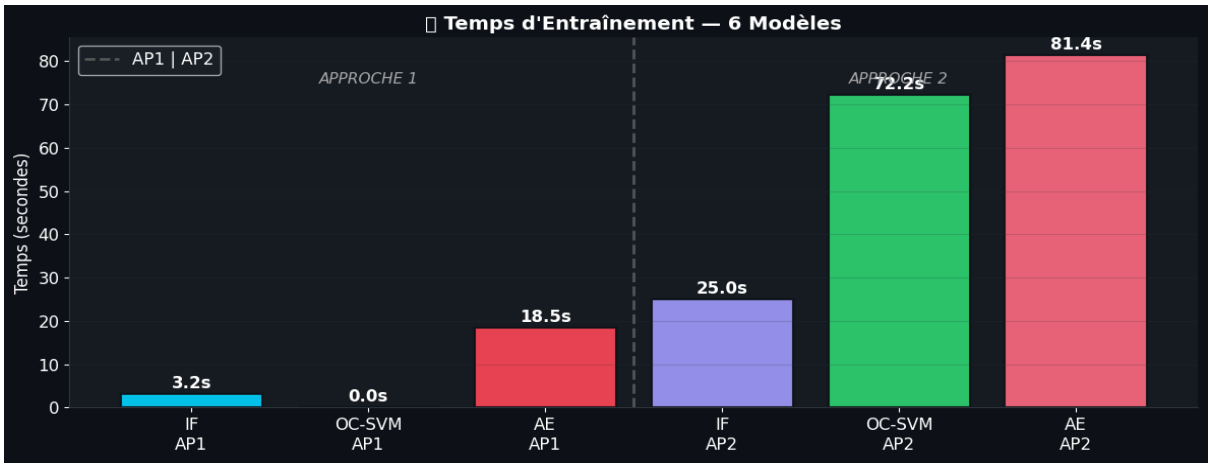


Figure 3. 8: Temps d'entraînement des six modèles en secondes.

3.7.2 Comparaison Approche 1 vs Approche 2 :

La figure 3.9 présente les courbes ROC (Receiver Operating Characteristic) obtenues pour six configurations expérimentales combinant trois algorithmes de détection d'anomalies (Isolation Forest (IF), One-Class SVM (OC-SVM) et Autoencoder (AE)) avec deux ensembles de caractéristiques (AP1 et AP2).

La courbe ROC représente le compromis entre :

TPR (True Positive Rate) : taux de détection des attaques (sensibilité).

FPR (False Positive Rate) : taux de fausses alertes.

La diagonale grise correspond à un classificateur aléatoire (AUC = 0,5). Plus une courbe est proche du coin supérieur gauche, meilleure est la performance du modèle. L'indicateur principal est l'AUC (Area Under the Curve) : plus elle est élevée, meilleure est la capacité de discrimination.

L'analyse comparative des deux approches met en évidence la supériorité globale de l'Approche 1 (one-class) par rapport à l'Approche 2 (semi-supervisée). En moyenne, l'Approche 1 atteint un F1-Score de 0,9914, contre 0,9126 pour l'Approche 2, représentant une amélioration de +0,0788. Par ailleurs, l'AUC-ROC moyen s'élève à 0,6498 pour l'Approche 1 contre 0,6250 pour l'Approche 2. Enfin, le temps d'entraînement total est significativement

réduit : 21,7 secondes pour l'Approche 1 contre 178,6 secondes pour l'Approche 2, faisant de l'approche one-class une méthode environ 8,2 fois plus rapide.

Ce résultat peut paraître contre-intuitif, étant donné que les modèles entraînés uniquement sur des données bénignes présentent de meilleures performances que ceux utilisant un jeu de données équilibré. Plusieurs facteurs expliquent ce phénomène. Tout d'abord, la forte proportion de malwares dans le jeu de données initial (89,2 %) complique l'apprentissage d'une représentation cohérente des comportements malveillants, d'autant plus qu'ils proviennent de diverses familles hétérogènes (Adware, Banking, SMS, Riskware). En revanche, les modèles one-class se concentrent exclusivement sur la distribution des comportements bénins, généralement plus homogènes et compactes. Ensuite, la génération d'échantillons synthétiques via la technique SMOTE peut introduire un biais artificiel dans la distribution des données, impactant la capacité de généralisation sur des données réelles.[40]

L'analyse des erreurs confirme cette tendance. Le meilleur modèle de l'Approche 1, IF AP1, ne produit aucun faux négatif, ce qui signifie qu'aucun malware n'a été manqué lors de la détection. En revanche, il génère 174 faux positifs, c'est-à-dire des applications bénignes identifiées à tort comme malveillantes. À l'inverse, IF AP2 engendre 54 faux négatifs et 156 faux positifs. Dans le contexte de la cybersécurité, les faux négatifs sont généralement considérés comme plus critiques que les faux positifs, car ils correspondent à des menaces non détectées pouvant compromettre le système. Ainsi, l'Approche 1 apparaît comme la stratégie la plus appropriée pour la détection d'anomalies dans le cadre de cette étude.

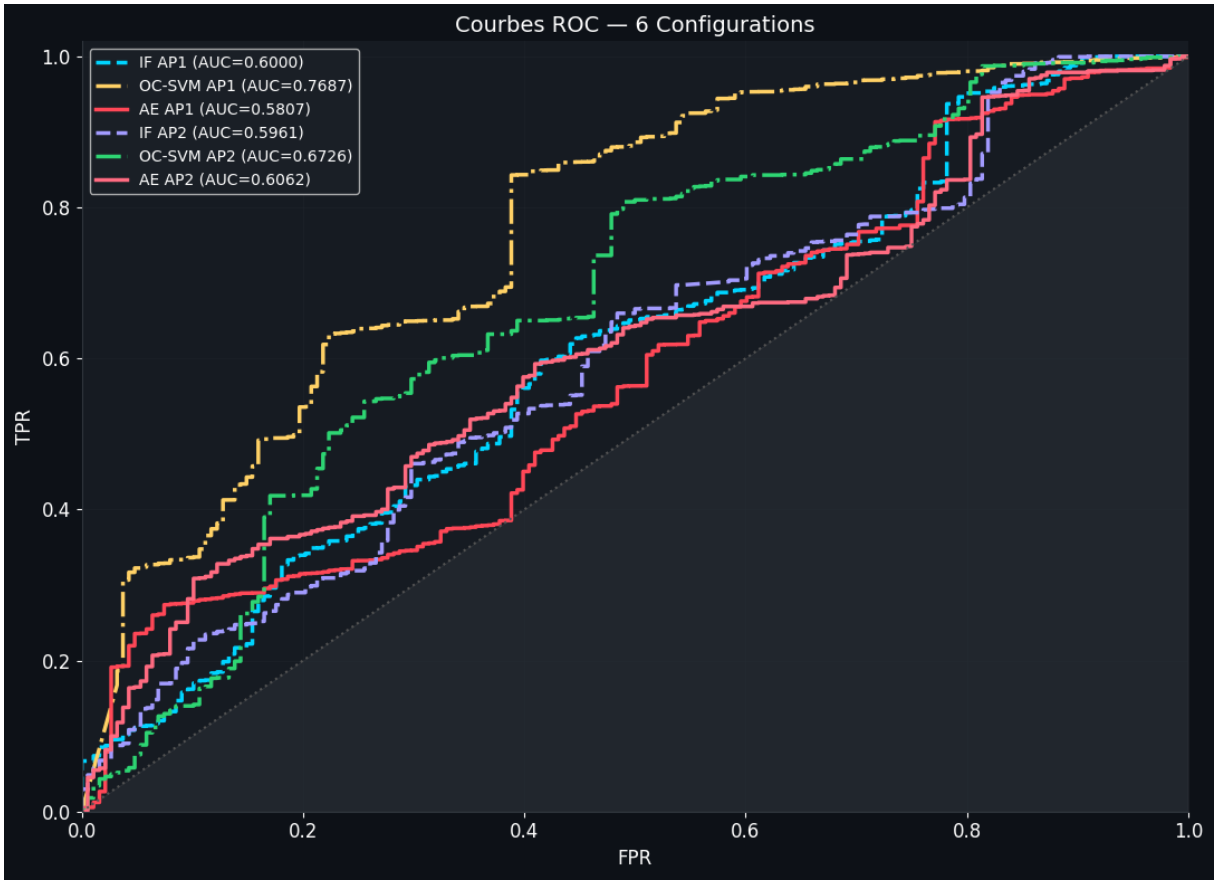


Figure 3. 9: Courbes ROC des six configurations expérimentales (IF, OC-SVM et AE) pour les approches AP1 et AP2.

3.7.3 Comparaison avec l'état de l'art :

Le tableau 3.10 présente une comparaison entre les résultats obtenus dans ce travail et plusieurs approches récentes appliquées au dataset CICMalDroid 2020. Les méthodes de référence utilisent principalement des approches supervisées nécessitant des exemples de malwares durant l'entraînement, comme Random Forest, SVM multi-classes ou des architectures profondes de type CNN-LSTM.

Travail	Méthode	Accuracy	F1-Score	Particularité
Rahali et al. (2021) [12]	Random Forest supervisé	97,80 %	0.977	Supervision complète
Kaluri et al. (2021) [45]	SVM multi-classes	94,50 %	0.943	Features statiques

Kim et al. (2022) [46]	CNN + LSTM	98,10 %	0.980	Features temporelles
Notre travail (IF AP1)	Isolation Forest one-class	98,35 %	0.9917	Sans labels malware
Notre travail (OC-SVM AP1)	One-Class SVM	98,22 %	0.9910	AUC=0.769

Tableau 3. 10 : Comparaison avec des travaux de l'état de l'art sur CICMalDroid 2020

Les résultats montrent que notre approche basée sur la détection d'anomalies obtient des performances très compétitives. Le modèle IF AP1 atteint une accuracy de 98,35 % ainsi qu'un F1-Score de 0,9917, dépassant les performances de plusieurs méthodes supervisées de l'état de l'art. De même, le modèle OC-SVM AP1 obtient une accuracy de 98,22 % avec un F1-Score de 0,9910 et une AUC-ROC de 0,769.

Ces résultats sont particulièrement importants car les modèles proposés ont été entraînés sans utiliser d'exemples de malwares. Contrairement aux approches supervisées classiques, les modèles one-class apprennent uniquement le comportement normal des applications bénignes puis détectent toute déviation comme une anomalie potentielle. Cette stratégie est particulièrement adaptée à la cybersécurité mobile et à la détection des attaques zero-day, c'est-à-dire des malwares nouveaux ou inconnus qui ne figurent pas dans les données d'apprentissage. Les méthodes supervisées peuvent en effet avoir des difficultés à reconnaître ces nouvelles attaques lorsqu'elles diffèrent des exemples vus durant l'entraînement [43].

3.8 DISCUSSION GENERALE ET LIMITES :

3.8.1 Forces de l'approche non supervisée :

L'approche en mode one-class proposée dans ce travail présente plusieurs avantages significatifs pour son application dans des environnements réels de cybersécurité mobile.

- Cette méthode ne repose pas sur l'utilisation de labels de logiciels malveillants lors de la phase d'entraînement. Le modèle apprend uniquement à partir d'applications bénignes, ce qui permet de réduire considérablement les coûts et le temps liés à l'annotation manuelle de logiciels malveillants, tâche souvent complexe et exigeante en ressources.

- Cette approche est particulièrement adaptée à la détection des attaques zero-day. En modélisant le comportement normal des applications légitimes, le système peut identifier comme anomalies des malwares inconnus ou non observés auparavant, contrairement aux méthodes basées sur des signatures ou des familles de malwares déjà recensées.

- Les modèles employés présentent une faible complexité computationnelle. L'algorithme Isolation Forest nécessite seulement 3,2 secondes pour l'entraînement, tandis que le One-Class SVM demande moins de 0,1 seconde, favorisant ainsi leur intégration dans des systèmes d'analyse performants et à grande échelle.

- L'intégration de l'analyse SHAP permet d'accroître l'interprétabilité des résultats. Cette méthode facilite l'identification de l'impact de chaque caractéristique sur la décision du modèle, contribuant à améliorer la transparence, la robustesse de l'audit ainsi que la validation scientifique du système de détection.

3.8.2 Limites et perspectives

Malgré les résultats prometteurs obtenus dans cette étude, il est essentiel de prendre en compte plusieurs limites afin d'évaluer de manière objective la portée des conclusions.

- Les expérimentations ont été réalisées uniquement sur le dataset CICMalDroid 2020. Dès lors, la capacité de généralisation des modèles à d'autres jeux de données Android, tels que DREBIN ou AndroZoo, n'a pas pu être validée. La réalisation d'évaluations croisées sur plusieurs jeux de données renforcerait la robustesse et la fiabilité des résultats.

- Les valeurs d'AUC-ROC observées restent relativement modérées (compris entre 0,58 et 0,77), ce qui indique que la distinction globale entre comportements bénins et malveillants demeure limitée. Cette situation pourrait impacter la stabilité des performances en fonction du seuil de décision choisi.

- L'évolution constante des malwares représente un défi majeur. Les malwares modernes, notamment ceux utilisant des techniques polymorphes ou de camouflage (packers), sont capables d'imiter des comportements bénins afin d'échapper aux systèmes de détection comportementale, ce qui peut compromettre l'efficacité des modèles proposés.

- L'approche 1 présente un taux élevé de faux positifs dans la configuration expérimentale adoptée. Bien que le modèle atteigne un rappel de 100 %, une proportion importante des applications bénignes de l'ensemble de test est classée à tort comme malveillante. Toutefois,

cette situation s'explique en partie par le fort déséquilibre de la configuration de test (188 applications bénignes contre 10 345 malwares). Dans un contexte plus équilibré, ce taux de faux positifs serait probablement inférieur.

Plusieurs axes d'amélioration sont envisageables pour de futures recherches. Parmi ceux-ci, l'utilisation de techniques d'apprentissage contrastif (contrastive learning) afin d'améliorer la qualité des représentations latentes, l'intégration de mécanismes d'attention pour identifier de manière dynamique les caractéristiques discriminantes, ainsi que l'exploration d'architectures Transformer adaptées aux données tabulaires.[44] Par ailleurs, la mise en place d'un système de réentraînement incrémental permettrait aux modèles de s'adapter progressivement à l'émergence de nouvelles familles de malwares.

3.9 CONCLUSION :

Ce chapitre a permis d'approfondir l'application des techniques de détection d'anomalies en cybersécurité mobile, à travers l'évaluation de trois modèles : Isolation Forest, One-Class SVM et Autoencodeur. Les expérimentations ont été menées sur le jeu de données CICMalDroid 2020, selon deux stratégies distinctes : une approche en mode one-class, basée exclusivement sur les applications bénignes, et une approche semi-supervisée reposant sur le rééquilibrage des classes par la méthode SMOTETomek.

Les résultats obtenus indiquent que l'approche one-class se révèle particulièrement adaptée dans le contexte analysé. Malgré un apprentissage effectué uniquement sur des comportements légitimes, les modèles ont démontré une capacité remarquable à détecter des applications malveillantes. Le modèle Isolation Forest, en particulier, a présenté des performances élevées en termes de précision, de rappel et d'indice F1, tout en maintenant un temps d'exécution raisonnable. Ces observations soulignent qu'un modèle capable d'apprendre efficacement le comportement normal peut identifier des anomalies sans nécessiter la connaissance préalable de toutes les variantes de malware lors de la phase d'entraînement.

L'étude a également mis en évidence les limites de l'approche semi-supervisée dans un contexte fortement déséquilibré, caractérisé par une grande diversité de familles malveillantes. Bien que les techniques de rééquilibrage telles que SMOTETomek améliorent la distribution des classes, elles peuvent aussi introduire des biais artificiels qui nuisent à la capacité de généralisation des modèles sur des données réelles. En revanche, les méthodes en mode one-

class apparaissent plus robustes face à l'hétérogénéité des menaces et mieux adaptées à la détection de malwares nouveaux ou inconnus.

Au-delà des performances quantitatives, cette recherche met en évidence l'intérêt pratique des méthodes étudiées pour des environnements Android en situation réelle. La faible charge computationnelle observée, notamment pour Isolation Forest et One-Class SVM, facilite leur intégration dans des systèmes de surveillance capables d'analyser rapidement de volumineuses collections d'applications. De plus, l'utilisation d'outils d'interprétabilité tels que SHAP contribue à une meilleure compréhension des décisions des modèles, un aspect essentiel pour renforcer la confiance et la traçabilité dans les systèmes de sécurité modernes.

Enfin, cette étude ouvre plusieurs perspectives pour de futurs travaux : l'évaluation sur d'autres jeux de données Android, l'intégration de modèles plus avancés, notamment basés sur les Transformers ou l'apprentissage contrastif, ainsi que le développement de systèmes adaptatifs capables d'évoluer face aux nouvelles générations de malwares.

CONCLUSION GENERALE :

La croissance rapide des smartphones et des applications mobiles a profondément transformé les usages numériques modernes. Cependant, cette évolution s'accompagne d'une augmentation considérable des menaces de cybersécurité, en particulier sur la plateforme Android, devenue une cible privilégiée des cybercriminels. Dans ce contexte, les approches traditionnelles de détection fondées sur les signatures et les règles statiques montrent aujourd'hui leurs limites face aux malwares polymorphes, aux techniques d'obfuscation et aux attaques zero-day.

Ce mémoire a ainsi proposé une approche de détection d'anomalies comportementales basée sur l'apprentissage automatique non supervisé, appliquée au dataset CICMalDroid 2020. L'objectif principal était d'évaluer la capacité de modèles entraînés exclusivement sur des comportements bénins à identifier efficacement des applications malveillantes inconnues, sans recours à des signatures préétablies.

Pour répondre à cet objectif, trois algorithmes complémentaires ont été étudiés : Isolation Forest, Autoencodeur et One-Class SVM. Deux protocoles expérimentaux ont été comparés : une approche one-class fondée uniquement sur des applications bénignes et une approche semi-supervisée intégrant un rééquilibrage SMOTETomek.

Les résultats expérimentaux ont mis en évidence la supériorité de l'approche one-class, notamment avec les modèles Isolation Forest et One-Class SVM, ayant atteint des F1-Scores supérieurs à 0,99. Ces performances démontrent l'efficacité des méthodes de détection comportementale pour identifier des menaces inconnues à partir de la modélisation du comportement normal des applications Android.

Ce travail souligne également l'intérêt des appels système (syscalls) comme représentation comportementale robuste et pertinente, ainsi que l'apport de l'analyse SHAP dans l'amélioration de l'interprétabilité et de la transparence des décisions prises par les modèles d'intelligence artificielle.

Malgré les résultats prometteurs obtenus, certaines limites demeurent, notamment l'utilisation d'un unique dataset expérimental et l'absence d'évaluation dans des environnements réels et dynamiques. De futures recherches pourraient ainsi explorer l'intégration d'architectures de deep learning plus avancées, l'apprentissage incrémental en

temps réel, ainsi que des approches multimodales combinant plusieurs sources de données comportementales.

En définitive, ce mémoire montre que l'apprentissage non supervisé représente une solution pertinente, performante et prometteuse pour la détection des malwares Android modernes. Cette approche ouvre des perspectives importantes pour le développement de systèmes de cybersécurité mobile plus adaptatifs, intelligents et capables de faire face à l'évolution constante des menaces numériques.

Références Bibliographiques

[1] Statista,” Number of Smartphone Users Worldwide”, Statista 2024 [Consulté le: Février 2026].

[2] StatCounter, "Mobile Operating System Market Share Worldwide," StatCounter GlobalStats, 2024. [Consulté le : Février 2026].

[3] D. Ehringer, "The Dalvik Virtual Machine Architecture," Université de Denver, Denver, CO, Rapport technique, 2010.

[4] W. Enck, M. Ongtang, et P. McDaniel, "Understanding Android Security," IEEE Security & Privacy, vol. 7, no. 1, pp. 50–57, Janv. /Févr. 2009.

[5] Mendix, "Mobile App Architecture," Mendix Documentation, 2024. [Consulté le : Février 2026].

[6] MDN Web Docs _Mozilla, "Progressive Web Apps," MDN Web Docs, 2024. [Consulté le: Février 2026].

[7] Flutter Documentation, "Architectural Overview," Flutter Docs, 2024. [Consulté le : Mars 2026].

[8] Android Developers _ Google, "Platform Architecture," Android Developers, 2024. [Consulté le : Février 2026].

[9] C. Miller, D. Blazakis, D. Dai Zovi, et al., iOS Hacker's Handbook. Hoboken : John Wiley & Sons, 2012.

[10] OWASP Foundation, "OWASP Mobile Application Security Verification Standard (MASVS) v2.0," OWASP, 2023. [Consulté le : Février 2026].

[11] Y. Ye, T. Li, D. Adjeroh, et S. S. Iyengar, "A Survey on Malware Detection Using Data Mining Techniques," ACM Computing Surveys, vol. 50, no. 3, Art. 41, pp. 1–40, Juin 2017.

[12] A. Rahali, E. B. Karbab, et M. Debbabi, "MalDroid 2020: A Large-Scale Benchmark Dataset for Android Malware Detection," Canadian Institute for Cybersecurity (CIC), Université du Nouveau-Brunswick, Rapport technique, 2020

[13] U. Bayer, P. M. Comparetti, C. Hlauschek, C. Kruegel, et E. Kirda, "Scalable, Behavior-Based Malware Clustering," dans Proc. 16th Annual Network and Distributed System Security Symposium (NDSS), San Diego, CA, USA, 2009, pp. 1–18.

[14] S. Arzt, S. Rasthofer, C. Fritz, et al., "FlowDroid: Precise Context, Flow, Field, Object-Sensitive and Lifecycle-Aware Taint Analysis for Android Apps," ACM SIGPLAN Notices, vol. 49, no. 6, pp. 259–269, Juin 2014

[15] D. E. Denning, "An Intrusion-Detection Model," IEEE Transactions on Software Engineering, vol. SE-13, no. 2, pp. 222–232, Févr. 1987.

[16] V. Chandola, A. Banerjee, et V. Kumar, "Anomaly Detection: A Survey," ACM Computing Surveys, vol. 41, no. 3, Art. 15, pp. 1–58, Juill. 2009.

[17] I. Burguera, U. Zurutuza, et S. Nadjm-Tehrani, "Crowdroid: Behavior-Based Malware Detection System for Android," dans Proc. 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM), Chicago, IL, USA, 2011, pp. 15–26.

[18] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, et Y. Weiss, "Android Malware Detection Based on Network Traffic Analysis," Applied Intelligence, vol. 38, no. 1, pp. 15–32, Janv. 2013.

[19] P. Faruki, A. Bharmal, V. Laxmi, et al., "Android Security: A Survey of Issues, Malware Penetration, and Defenses," IEEE Communications Surveys & Tutorials, vol. 17, no. 2, pp. 998–1022, 2015.

[20] N. Idika et A. P. Mathur, "A Survey of Malware Detection Techniques," Purdue University, West Lafayette, IN, CS Technical Report, 2007.

[21] I. You et K. Yim, "Malware Obfuscation Techniques: A Brief Survey," dans Proc. 2010 International Conference on Broadband, Wireless Computing, Communication and Applications (BWCCA), Fukuoka, Japon, 2010, pp. 297–300.

[22] N. Milosevic, A. Dehghantanha, et K.-K. R. Choo, "Machine Learning Aided Android Malware Classification," Computers & Electrical Engineering, vol. 61, pp. 266–274, Juill. 2017.

[23] Tom. M. Mitchell, Machine Learning. New York : McGraw-Hill Science/Engineering/Math, 1997, pp. 1–432.

[24] I. Goodfellow, Y. Bengio, et A. Courville, Deep Learning. Cambridge : MIT Press, 2016, pp. 1–800.

[25] M. M. Breunig, H.-P. Kriegel, R. T. Ng, et J. Sander, "LOF: Identifying Density-Based Local Outliers," dans Proc. ACM SIGMOD International Conference on Management of Data, Dallas, TX, USA, 2000, pp. 93–104.

[26] Scikit-learn Developers, "User Guide: Novelty and Outlier Detection," scikit-learn Documentation, 2024. [Consulté le : avril 2026].

[27] Shahroz Tariq et al., "CICMalDroid 2020: Android Malware Detection Dataset," Canadian Institute for Cybersecurity, Université du Nouveau-Brunswick, 2020 [Consulté le : avril 2026].

[28] F. T. Liu, K. M. Ting, et Z.-H. Zhou, "Isolation Forest," dans Proc. 8th IEEE International Conference on Data Mining (ICDM), Pisa, Italie, 15–19 Déc. 2008, pp. 413–422.

[29] J. Schmidhuber, "Deep Learning in Neural Networks: An Overview," Neural Networks, vol. 61, pp. 85–117, Janv. 2015

[30] M. K. Alzaylaee, S. Y. Yerima, et S. Sezer, "DL-Droid: Deep Learning Based Android Malware Detection Using Real Devices," Computers & Security, vol. 89, Art. 101663, pp. 1–17, Févr. 2020.

[31] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, et R. C. Williamson, "Estimating the Support of a High-Dimensional Distribution," Neural Computation, vol. 13, no. 7, pp. 1443–1471, Juill. 2001.

[32] Kaggle Inc., "Kaggle Notebooks: A Collaborative Data Science Platform," Kaggle, 2024. [Consulté le : Février 2026].

[33] G. Van Rossum et F. L. Drake, Python 3 Reference Manual. Scotts Valley : CreateSpace, 2009, pp. 1–242.

[34] G. Dini, F. Martinelli, A. Saracino, et D. Sgandurra, "MADAM: A Multi-Level Anomaly Detector for Android Malwares," dans Computer Network Security, Lecture Notes in Computer Science, vol. 7531. Berlin : Springer, 2012, pp. 240–253.

[35] K. Tam, S. J. Khan, A. Fattori, et L. Cavallaro, "CopperDroid: Automatic Reconstruction of Android Malware Behaviors," dans Proc. ISOC Network and Distributed System Security Symposium (NDSS), San Diego, CA, USA, 2015, pp. 1–15.

[36] S. Hariri, M. C. Kind, et R. J. Brunner, "Extended Isolation Forest," IEEE Transactions on Knowledge and Data Engineering, vol. 33, no. 4, pp. 1479–1489, Avr. 2021.

[37] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, et R. C. Williamson, "Estimating the Support of a High-Dimensional Distribution," Microsoft Research, Redmond, WA, Rapport technique MSR-TR-99-87, 1999.

[38] G. E. Hinton et R. R. Salakhutdinov, "Reducing the Dimensionality of Data with Neural Networks," Science, vol. 313, no. 5786, pp. 504–507, Juill. 2006.

[39] S. Kaufman, S. Rosset, C. Perlich, et O. Stitelman, "Leakage in Data Mining: Formulation, Detection, and Avoidance," ACM Transactions on Knowledge Discovery from Data, vol. 6, no. 4, Art. 15, pp. 1–21, Déc. 2012.

[40] N. V. Chawla, K. W. Bowyer, L. O. Hall, et W. P. Kegelmeyer, "SMOTE: Synthetic Minority Over-Sampling Technique," Journal of Artificial Intelligence Research, vol. 16, pp. 321–357, Juin 2002.

[41] B. Schölkopf et A. J. Smola, Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond. Cambridge : MIT Press, 2002, pp. 1–644.

[42] G. E. Batista, R. C. Prati, et M. C. Monard, "A Study of the Behavior of Several Methods for Balancing Machine Learning Training Data," ACM SIGKDD Explorations Newsletter, vol. 6, no. 1, pp. 20–29, Juin 2004.

[43] M. Salehi, T. Mirzaei, et S. Ghiasi, "A Survey on Anomaly Detection in Cybersecurity: Machine Learning Approaches," ACM Computing Surveys, vol. 54, no. 2, Art. 37, pp. 1–36, Mars 2022

[44] L. Grinsztajn, E. Oyallon, et G. Varoquaux, "Why Tree-Based Models Still Outperform Deep Learning on Tabular Data," dans Advances in Neural Information Processing Systems (NeurIPS 2022), vol. 35, pp. 507–520, 2022.

[45] R. Kaluri et al., "Support Vector Machine-Based Android Malware Detection Using Static Analysis," International Journal of Advanced Computer Science and Applications, vol. 12, no. 3, pp. 123–131, 2021.

[46] J. Kim et J. Kim, "CNN-LSTM Based Malware Detection Using Behavioral Features from Android Applications," IEEE Access, vol. 10, pp. 31902–31913, 2022.

RÉSUMÉ

Android s'est imposé comme le système d'exploitation mobile le plus répandu, faisant de cette plateforme une cible privilégiée pour les cybercriminels. Face à la croissance des logiciels malveillants ciblant Android, les techniques de détection classiques basées sur les signatures montrent leurs limites. Ce travail propose une approche de détection d'anomalies par intelligence artificielle, combinant trois modèles non supervisés — Isolation Forest, One-Class SVM et Autoencodeur — appliqués au dataset CICMalDroid 2020, dans le but d'identifier efficacement les applications malveillantes grâce à leur capacité d'adaptation aux menaces inédites.

Mots clés : Android, détection d'anomalies, malware, apprentissage non supervisé, Autoencodeur, One-Class SVM, Isolation Forest, CICMalDroid 2020.

ABSTRACT

Android has established itself as the most widely used mobile operating system, making it a prime target for cybercriminals. Faced with the growing threat of Android malware, conventional signature-based detection techniques show their limitations. This work proposes an AI-based anomaly detection approach combining three unsupervised models — Isolation Forest, One-Class SVM, and Autoencoder — applied to the CICMalDroid 2020 dataset, aiming to effectively identify malicious applications through their adaptive capabilities against novel threats.

Keywords: Android, anomaly detection, malware, unsupervised learning, Autoencoder, One-Class SVM, Isolation Forest, CICMalDroid 2020.

ملخص

أصبح نظام اندرويد الأكثر انتشارا على مستوى العالم، مما جعله هدفا رئيسيا لمجرمي الإنترنت. في مواجهة التزايد المستمر للبرمجيات الخبيثة، تظهر تقنيات الكشف التقليدية محدوديتها. يقترح هذا العمل نهجا للكشف عن الاختلالات في تطبيقات أندرويد باستخدام الذكاء الاصطناعي، يجمع بين ثلاثة نماذج Isolation Forest و Autoencodeur و One-Class SVM مطبقة على مجموعة البينات CICMalDroid 2020 بهدف التعرف الفعال على التطبيقات الخبيثة بفضل قدرتها على التكيف مع التهديدات غير المعروفة.

الكلمات المفتاحية : كشف الحالات غير الطبيعية ، أندرويد ، برمجيات خبيثة ، تعلم غير خاضع للإشراف ، Isolation Forest، Autoencodeur ، One-Class SVM ، CICMalDroid 2020 .